ABC: Abstract prediction Before Concreteness

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

Man-Ki Yoon
Computer Science
Yale University
New Haven, CT, USA
man-ki.yoon@yale.edu

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

***Abstract*—Learning techniques are advancing the utility and capability of modern embedded systems. However, the challenge of incorporating learning modules into embedded systems is that computing resources are scarce. For such a resource-constrained environment, we have developed a framework for learning abstract information early and learning more concretely as time allows. The intermediate results can be utilized to prepare for early decisions/actions as needed. To apply this framework to a classification task, the datasets are categorized in an abstraction hierarchy. Then the framework classifies intermediate labels from the most abstract level to the most concrete. Our proposed method outperforms the existing approaches and reference baselines in terms of accuracy. We show our framework with different architectures and on various benchmark datasets CIFAR-10, CIFAR-100, and GTSRB. We measure prediction times on GPU-equipped embedded computing platforms as well.**

***Index Terms*—adaptive concreteness, resource-constrained system, cyber-physical system, adaptive neural network**

I. Introduction

Learning techniques have been advancing in many areas and contexts of modern computing, and embedded systems are no exception. The challenge of incorporating a learning module into an embedded system is that resources, such as size, power, time, memory, etc., are not abundant, unlike in general purpose systems. However, in the literature of learning techniques, such a resource-constrained operating environment has not received much attention. With this motivation in mind, we propose a framework that enables a classification module to obtain information at different abstraction levels given a constrained resource, in particular, *time*.

To take a well-studied case, a classifier system takes input and determines which of several classes most likely correspond to the given input. In the real-time embedded context, such a system can provide benefit by obtaining an abstracted result quickly and continuing to work toward computing a final answer as long as time permits. For example, as shown in Fig. 1, recognizing a category that contains a "stop" sign (*i.e.*, urgent signs) is more time-critical than one containing "speed limit" signs. This is because a stop sign requires early action in a timely manner.

To support this functionality, the input data can be *hierarchically* categorized like the example shown in Fig. 2. In fact, CNN-based architectures or ResNet-based architectures (which also include multiple convolutional layers, [1]) are hierarchical, in that lower-level features are extracted first, and higher-level features are obtained later. Therefore, such models can naturally be adapted to the problem of hierarchical classification.

The proposed architectural framework not only realizes this functionality, but also enhances overall classification accuracy. The schematic description in Fig. 3 shows our framework, which enables multi-level classification. The key idea is the introduction of "Classification Feature Feedback" (CFF) which feeds back each level's classification feature to the main flow. This functionality enables a later-level classification to build on the previous classification knowledge. In addition, the knowledge from the intermediate classifications is also incorporated into losses - the local losses are aggregated with different weights into the global loss.

With CNN and ResNet architectures, we show that the proposed framework consistently improves the overall accuracy performance across multiple datasets. We also measure prediction times on GPU-equipped embedded computing platforms and demonstrate the feasibility and practicality of ABC.

II. Network Architecture for Abstract prediction Before Concreteness (ABC)

To facilitate the production of intermediate classifications, we categorize the input data in an abstraction hierarchy as shown in Fig. 2. The network provides abstract (coarser) classifications earlier before producing the final concrete classification, as the framework's name suggests, ABC (Abstract prediction Before Concreteness). Hence, as shown in Fig. 3, each level produces input refinements that are fed into the ongoing feed-forwarding process. The total number of classification levels in the hierarchy corresponds to the number of levels in the hierarchy of the target dataset.

Any layered neural network architecture can implement the ABC architectural skeleton and provide hierarchical classification results. In this paper, we study two architectures that implement ABC; one uses basic CNN components (for a shallower neural network) and the other uses ResNet blocks (for a deeper neural network). Although both architectures are convolutional neural networks sharing many similarities, we also consider a ResNet-based architecture because of their current prominence in the literature and frequent usage in image-processing applications.

This work is supported in part by grants from NSF 1521523 and 1715154. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of sponsors.

Fig. 1: Conceptual overview of hierarchical classification task.

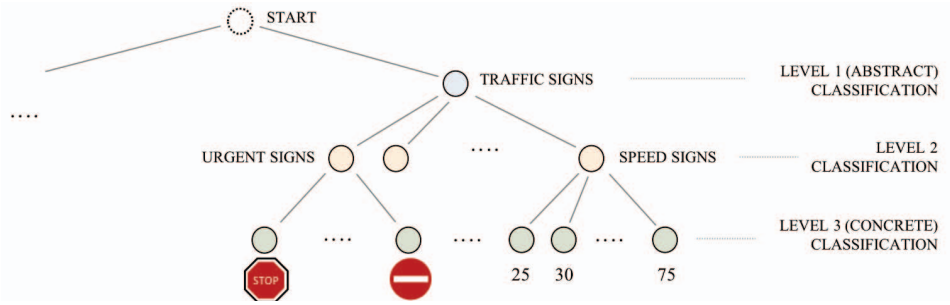


Fig. 2: Example abstraction hierarchy.

A. Classification Feature Feedback (CFF)

One of the contributions of our framework is *Classification Feature Feedback* (CFF) which *reincorporates* the previous classification features to the remainder of the network, represented as red flows in Fig. 3. CFF enables a later level classification to build on top of the previous classification knowledge. Formally, let $H_1, H_2, \cdots, H_i$ be transformations of the layers on the branch for a classification level, and let $H = H_i \circ H_{i-1} \circ \cdots \circ H_1$. Afterwards, CFF forwards $x + H(x)$ to the next level rather than just x. Effectively, CFF forms something similar to the "shortcut connection" used in ResNet [1]. Reincorporating each level's classification features, $H(x)$, enhances the performance significantly. Although there exists CNN-based architectures for hierarchical classification [2, 3, 4], they do not incorporate intermediate classification features into the next levels as our ABC does. Through experiments, we compare the performance of both architectures, with and without CFF, and show that CFF consistently improves performance of the final classification.

B. Loss Protocols

The loss function L for ABC models consists of a weighted sum of the component losses,

$$L = \sum_{n=1}^{N} W_n L_n, \tag{1}$$

where each L_n is the corresponding loss at classification level n, and W_n is the corresponding weight. The precise structure of each L_n is general, but for the purposes of our experiments, we use the standard categorical cross-entropy loss.

Essential to final model performance is the weighting *schedule* – how do we choose each W_n, and how do we vary the the weights across the training procedure? A full exploration of such questions is outside the scope of this paper, but in attempting to empirically determine a fair schedule,

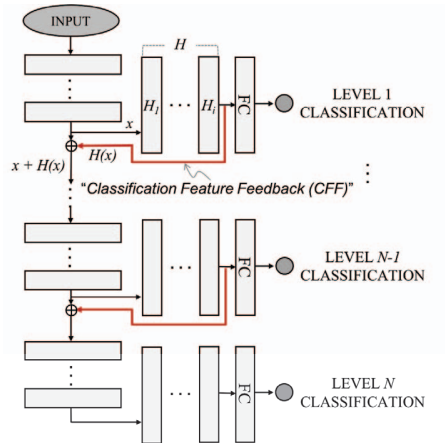
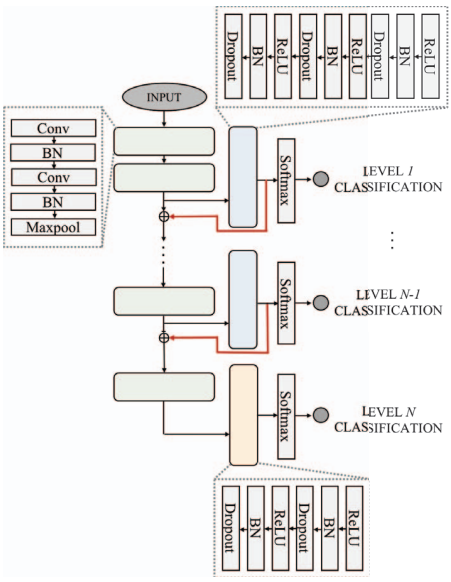


Fig. 3: Overview of the proposed framework.

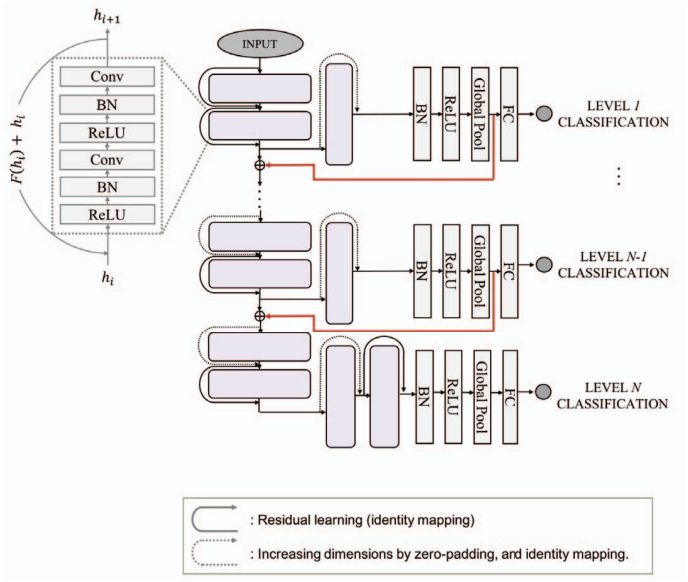
we noted that static weighting schemes (*e.g.*, letting a be a constant, setting $W_n = a$ for all n) produced universally poor performance across models. More details are presented in Sec. III-A.

C. Case Study 1: CNN-based architecture

The first implementation is a generic CNN-structure, showcasing the generalizability of the ABC framework. The overview of the CNN architecture is described in Fig. 4(a). The skeleton of our CNN network is a slightly-modified version of the network presented in [3, 5]. The main part of the network consists of repeated convolution and batch normalization layers, and the exit branches are comprised of densely-connected layers followed by batch normalization and dropout as shown in Fig. 4(a). The convolutions in the network begin with a feature size of 64, and double a total of 3 times, resulting in a feature size of 512 in the final convolution layers. The final convolution is then flattened and passed over to a dense block



(a) CNN-based architecture.



(b) ResNet-based architecture.

Fig. 4: Architecture overviews

consisting of three dense layers, two of which are dimension 1024, and the final dense layer has dimension equal to the number of fine prediction classes.

The early exit branches have a similar structure to the final dense block described above: three dense layers, the last of which has dimension equal to the number of (early) classes. To implement CFF, we perform dimensionality expansion on the dropout layer preceding the branch's final densely-connected layer, and add the resulting tensor to the input of the following convolution layer.

D. Case Study 2: ResNet-based architecture

The base architecture of our second implementation instance is an 18-layer ResNet architecture as described in [1]. ResNets are deep convolutional networks [1, 6] employing residual connections, *i.e.*, letting $\mathbf{h}_i$ represent hidden layer i and F represent a transformation function,

$$\mathbf{h}_{i+1} = \mathbf{h}_i + F(\mathbf{h}_i) \tag{2}$$

Each residual block F is primarily characterized through the specification ($m \times m$ kernel size and number of filters k) of its convolution operation. The layer configuration of a single ResNet block can be found in Fig. 4(b) which is ReLU→BatchNorm→Conv→ReLU→BatchNorm→Conv.

As noted earlier, our framework reincorporates the information derived from the previous classification. The structure of the early exit branches mirrors that of the ResNet itself, consisting of a ResNet block, followed by an exit block consisting of BatchNorm→ReLU→Global Pooling→Dense. Similar to the CNN network, we reincorporate the classification features back into the feed-forward portion of the network by summing them with the input to the next block (see the red arrows in Fig. 4(b)).

III. Experiments

In this section, we examine the performance of the CNN and ResNet models using the proposed ABC framework. We examine both accuracy and timing metrics.

A. Evaluation Setup

All of the models presented in this section are trained 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).

1) Data Sets: We evaluate our models over three datasets:

- **CIFAR-10**: The original CIFAR-10 [7] 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. The images are classified according to one of ten classes. We created a hierarchical structure as in [3], creating a top-level distinction between "TRANSPORT" and "ANIMAL" images, and a secondary level classification problem consisting of 7 classes (SKY, WATER and ROAD for TRANSPORT, and BIRD, REPTILE, PET and MEDIUM-SIZED for ANIMAL). The concrete level classes correspond to the original classes.

TABLE I: CIFAR-100 class mapping

Metaclass	CIFAR-100 Superclasses
AQUATIC ANIMALS	AQUATIC MAMMALS, FISH
PLANTS	FLOWERS, FRUITS AND VEGETABLES, TREES
HOUSEHOLD ITEMS	FOOD CONTAINERS, HOUSEHOLD FURNITURE, HOUSEHOLD ELECTRICAL DEVICES
INVERTEBRATES	INSECTS, NON-INSECT INVERTEBRATES
ANIMALS	LARGE OMNIVORES AND HERBIVORES, REPTILES, MEDIUM-SIZED MAMMALS, SMALL-SIZED MAMMALS, LARGE CARNIVORES
OUTDOOR OBJECTS	LARGE MAN-MADE OUTDOOR THINGS, LARGE NATURAL OUTDOOR THINGS
PEOPLE	PEOPLE
VEHICLES	VEHICLES 1, VEHICLES 2

TABLE II: Weight schedule for both architectures

Epoch	W_1	W_2	W_3
0	0.98	0.01	0.01
10	0.1	0.8	0.1
20	0.1	0.2	0.7
30	0.05	0.15	0.85

- **CIFAR-100**: The CIFAR-100 [7] 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. The images are classified as one of 100 classes. In addition to the 100 classes, a coarse set of labels is provided that groups the 100 original classes into a set of 20 metaclasses. We provide one additional coarse layer that groups the 20 coarse classes into 8 classes. We specify this mapping of coarse labels to "extra coarse" labels in Table I.
- **GTSRB**: The GTSRB [8] dataset consists of a pre-divided training and test dataset, covering 43 classes of traffic signs. The training and test datasets contain 39,209 and 12,630 images, respectively. For a hierarchical structure, the 43 classes are abstracted into 6 higher classes (SPEED LIMIT, OTHER PROHIBITORY, DERESTRICTION, MANDATORY, DANGER, and UNIQUE (PRIORITY) SIGNS), and then we again abstracted the 6 classes into 2 metaclasses for a higher level.

We perform simple data pre-processing, subtracting the mean training image from the training and test set, and scaling the pixel-values between $[-1, 1]$. We augmented the training data by randomly shifting images vertically and horizontally (by a factor of 0.1 of the total height/width) and performing random horizontal flips.

B. CNN-based Architecture

1) Approaches, implementations, hyperparameters: We use the following three architectures for our analysis:

- FCBASE (FlatCNNBase): As a reference architecture to compare with, we implemented FLATBASE which is a non-hierarchical, feed-forward CNN-based architecture. It contains eight convolutional layers and three densely connected layers.
- HCBASE (HierarchicalCNNBase): Based on the FCBASE architecture, with the addition of two early prediction branches. Equivalent to the architecture in Fig. 4(a), without the red CFF connections.
- HCOURS (HierarchicalCNNOurs): The architecture proposed in this paper. At each of the two intermediate levels, classification features are fed back to the feed-forward flow. This approach implements CFF – please refer to Fig. 4(a) for details.

We use stochastic gradient descent with momentum $\mu = 0.9$ to train all of our models. The weight schedules for each dataset are given in Table II, and the learning rate decay schedules are given in Table III.

2) Performance: The results are shown for each of the three datasets. Across datasets, the consistent pattern that we need to notice is that our proposed HCOURS clearly outperforms HCBASE by the final-level classification. The impact of CFF itself is more directly shown by comparing HCBASE and HCOURS. We attribute this to the impact of CFF which keeps feeding intermediate classification features to the final level.

TABLE III: Learning rate schedule

	CNN		ResNet	
	Epoch	Learning rate	Epoch	Learning rate
CIFAR-10	0	0.003	0	0.003
	40	0.0005	60	0.0005
	50	0.0001	80	0.0001
CIFAR-100 / GTSRB	0	0.001	0	0.01
	55	0.0002	60	0.001
	70	0.00005	80	0.0001

TABLE IV: Test accuracy in CNN-based architectures.

		Level 1 (abstract)	Level 2	Level 3 (concrete)
CIFAR-10	FCBASE	–	–	82.60 %
	HCBASE	95.48 %	87.13 %	83.54 %
	HCOURS	96.03 %	86.76 %	**84.38** %
CIFAR-100	FCBASE	–	–	50.26 %
	HCBASE	68.67 %	60.45 %	55.55 %
	HCOURS	69.09%	58.37 %	**56.43** %
GTSRB	FCBASE	–	–	93.97 %
	HCBASE	99.24 %	99.17 %	95.03 %
	HCOURS	99.30 %	99.53 %	**97.24** %

This suggests that given a hierarchical dataset and a goal of accuracy in the final prediction (i.e., the user exclusively cares about the final prediction), then the CFF architecture should be chosen without reservation. Even if the classification target is concerned with intermediate classifications, the CFF architecture, HCOURS, is still superior for the final accuracy. Although the accuracies for GTSRB are already high enough, our CFF architecture still shows consistent performance improvement for the final accuracy.

However, we find that the earlier predictions are sensitive to the loss weight schedule. Indeed, an early weight schedule we used set $W_0 = W_1 = 0$ and $W_2 = 1$, meaning only the final prediction contributed to the loss by the end of training. This had a minor negative impact on the early branches of HCBASE, but a significantly negative impact on the early branches of HCOurs. Adequate time must be spent tuning those hyperparameters to ensure competitive performance of the early branches of HCOurs.

One other pattern is that for any hierarchical architecture/dataset, result accuracy is better in abstract classification than in concrete classification. This is intuitive, since for abstract classification a datapoint falls into one of a smaller number of cases than in the concrete case, creating a probabilistically easier problem.

3) Timing: We performed timing measurements for our proposed method as well as the other baseline architectures on NVIDIA Jetson TX2 [9], an embedded computing device targeting mainly for mobile robot applications such as the miniaturized self-driving car shown in Fig. 1. It features a 256-core NVIDIA Pascal GPU (1.3 tera operations per second), which we configured to operate at the maximum frequency.

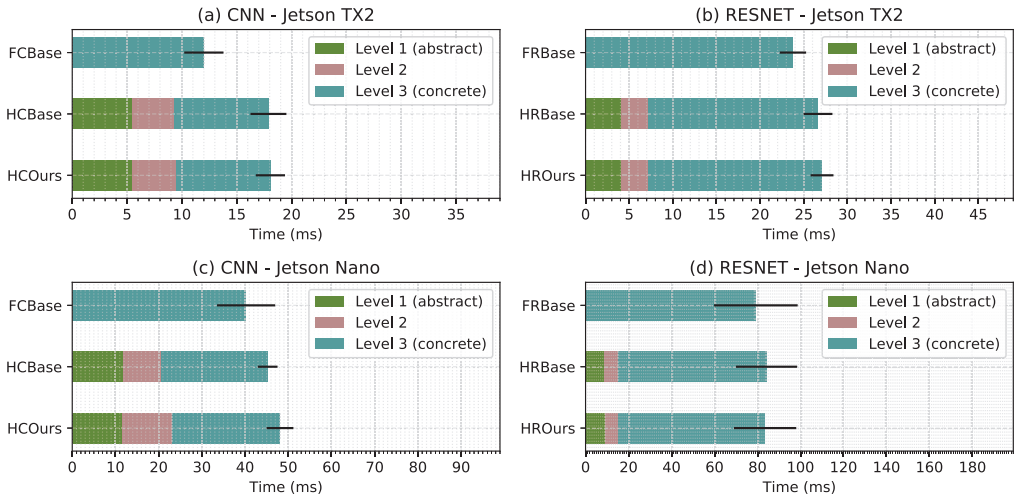


Fig. 5: Average prediction times for different classification levels of CIFAR-10 class in CNN (left) and ResNet (right) architectures on NVIDIA Jetson TX2 (top) and Nano (bottom). Notice the different time scales (*i.e.*, x-axis) of the plots.

We measured the times that each architecture takes to output the predicted class at each level (Levels 1, 2, and 3), as shown in Fig. 5(a), using the 10,000 test images of CIFAR-10. Recall that FCBASE outputs only the final-level class.

The hierarchical architecture (HCOURS and HCBASE) can make (highly accurate) predictions on Levels 1 and 2 within 10 ms on average, during which the non-hierarchical FCBASE cannot output any result. This demonstrates the capability of ABC to enable intermediate prediction. We also observe that the difference between HCBASE and HCOURS is negligible, indicating that our method does not incur temporal overhead compared to the baseline. We performed the same experiments on NVIDIA Jetson Nano [10], a lower-end device than TX2, and observed the same trends (except 2–3 times of increase in prediction times due to the less-powerful GPU of Nano) as shown in Fig. 5(c).

C. ResNet-based Architecture

1) Approaches, implementations, hyperparameters: Below are the models used for our ResNet-based implementation:

- FRBASE (FlatResNetBase): As a reference to compare with, we implemented FRBASE which is a non-hierarchical feed-forward ResNet-based architecture. It is a ResNet-18, containing 17 convolutional layers and 1 dense layer.
- HRBASE (HierarchicalResNetBase): Based on the FRBASE architecture, with the addition of two early prediction branches. Equivalent to the architecture in Fig. 4(b), without the CFF connections.
- HROURS (HierarchicalResNetOurs): The architecture proposed in this paper which implements CFF. That is, at each of the two intermediate levels, classification features are fed back to the feed-forward flow. Please refer to Fig. 4(b).

The weight schedule is the same as given in Table II, and the learning rate schedule is given in table III.

TABLE V: Test accuracy in ResNet-based architectures.

		Level 1 (abstract)	Level 2	Level 3 (concrete)
CIFAR-10	FRBASE	–	–	85.83 %
	HRBASE	95.08 %	86.98 %	86.08 %
	HROURS	95.24 %	87.01 %	**86.45** %
CIFAR-100	FRBASE	–	–	60.21 %
	HRBASE	78.71 %	72.77%	61.88 %
	HROURS	75.81 %	71.23%	**62.94** %
GTSRB	FRBASE	–	–	94.12 %
	HRBASE	99.60 %	99.55 %	94.40 %
	HROURS	99.51 %	99.60 %	**95.53** %

2) Performance: The performance of the ResNet variants is shown in Table V. Although learning rate schedules differ from the CNN case, consistent results and trends are again found in ResNet architectures. Similar to the CNN case, we compare three architectures, a non-hierarchical baseline, FRBASE, a hierarchical baseline, HRBASE, and our hierarchical network, HROURS. Across all datasets, HROURS with CFF connections consistently outperforms the other architectures by the final-level classification. As we noted in the CNN case, in the final level accuracy the gap between HRBASE and HROURS shows the impact of CFF connections. We also find that abstract classification shows better performance than concrete one, since abstract classification deals with an easier problem.

3) Timing: As done in Section III-B3, we performed timing measurements with the ResNet implementation. The results presented in Fig. 5(b) highlight further the merit of the hierarchical approaches; both HRBASE and HROURS take only about 13% longer than FRBASE to predict the first-level class while the first and second level classes can be obtained after spending only 15% and 26%, respectively, of the entire execution time. Performing the same experiments on Jetson Nano (shown in Fig. 5(d)) lowered these numbers to 10% and 17%, respectively, which highlights the benefit of the

intermediate classification capability that ABC provides.

IV. Related Work

For hierarchical classification, previous work has focused on how to generate a hierarchy (*i.e.*, hierarchical categories) for a dataset through some learning mechanism [4, 11, 12], or on how to enhance performance, but has not addressed exploiting intermediate results [2, 3]. In a hierarchical dataset's label tree, each non-leaf node has a corresponding separate network in [13]. Although intuitive, this may not scale well since the model must have the same number of individual classifiers as the number of non-leaf nodes.

The ResNet (residual net) architecture, characterized by skip connections/identity mappings, was introduced in [1] and was a breakthrough in deep learning. Since then, many variants and optimizations have been explored in the literature including [14, 15]. A further branch of work explores ResNet's mechanism of action – one school of thought claims the performance of ResNet is largely due to its tendency to behave like an ensemble of shallower networks [16, 17], while others claim ResNet's performance is due to its tendency to iteratively refine features [18]. Thanks to the modularity of ResNets, ResNet building blocks can be dropped/shuffled without significantly impacting the network's performance [19]. Aside from performance enhancements, the tendency of ResNet to iteratively improve features makes it ideal for obtaining intermediate results.

The idea of neural networks with early exits has been explored in the literature including [20, 21]. Most of these early-exit approaches skip layers automatically if the result are under certain confidence thresholds midway through the computation, just in order to enhance computational efficiency,. This kind of architectures are well-suited to domains having restrictions on computational costs during inference, allowing the user to exit earlier with a coarse-grained prediction, or expend more time and computational power to retrieve a finer-grained prediction.

Attention methods [22, 23] first focuses on portions of input data that are most significant to the final classification result. Their applications are found in the literature such as [24, 25]. However,, attention falls short of solving the problem of hierarchical prediction, and implementing an attention mechanism takes many more parameters that need to be trained and increases the computational effort.

V. Conclusion

In this paper, we propose an architectural framework that provides abstract classification information quickly and provides concrete results later as time allows. Classification feature feedback improves performance in hierarchical neural networks. In addition, we explored weighting schemes for losses from intermediate classifications for overall performance enhancement. Finally, our experiments demonstrate the benefit of ABC in providing coarse but valid information far ahead of when a non-hierarchical baseline does. Finding an optimal weighting scheme for intermediate losses will be one of the directions going forward to improve the overall performance.

References

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

[2] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. Hd-cnn: Hierarchical deep convolutional neural networks for large scale visual recognition. In *IEEE ICCV*, pages 2740–2748, 2015.

[3] Xinqi Zhu and Michael Bain. B-CNN: branch convolutional neural network for hierarchical classification. *CoRR*, abs/1709.09890, 2017.

[4] Yanming Guo, Yu Liu, Erwin M. Bakker, Yuanhao Guo, and Michael S. Lew. CNN-RNN: a large-scale hierarchical image classification framework. *Multimedia Tools Appl.*, 77(8):10251–10271, 2018.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

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

[8] 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.

[9] NVIDIA. Jetson TX2 Module. https://developer.nvidia.com/embedded/buy/jetson-tx2.

[10] NVIDIA. Jetson Nano developer kit. https://developer.nvidia.com/embedded/buy/jetson-nano-devkit.

[11] Hichem Bannour and Céline Hudelot. Hierarchical image annotation using semantic hierarchies. In *ACM CIKM*, pages 2431–2434, 2012.

[12] Ruslan Salakhutdinov, Antonio Torralba, and Joshua B. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, pages 1481–1488, 2011.

[13] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. Tree-cnn: A deep convolutional neural network for lifelong learning. *CoRR*, 2018.

[14] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *IEEE CVPR*, pages 5987–5995, 2017.

[15] Ohad Shamir. Are resnets provably better than linear predictors? In *NIPS*, pages 507–516, 2018.

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

[17] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *ICLR*, 2017.

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

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

[20] 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.

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

[22] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *NIPS*, pages 1243–1251. 2010.

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

[24] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057, 2015.

[25] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.