Adapting branched networks to realise progressive intelligence

Jack Dymond
j.dymond@soton.ac.uk
Sebastian Stein
ss2@soton.ac.uk
Steve R. Gunn
srg@soton.ac.uk

Electronics and Computer Science
University of Southampton
Southampton
United Kingdom

Abstract

Progressive intelligence is a formulation of machine learning which trades-off performance requirements with resource availability. It does this by approaching the inference process incrementally. Current work in this area focuses on overall model performance rather than optimising its complete operating range. In this paper, we build upon existing explainability and branched neural network research to show how neural networks can be adapted to exhibit progressive intelligence.

We assess the utility of joint branch optimisation for progressive intelligence using a number of explainability metrics. When optimising the area under curve of layer-wise linear probe accuracy we find equally weighted early-exit branch optimisation produces models with the highest linear probe accuracy throughout the backbone. By varying confidence thresholds we represent the entire range over which the model can operate, we then explore its interaction with the scaling of the branched neural network backbone. Finally, we propose a novel ensemble inference strategy which utilises repeat predictions and requires no additional optimisation. Experiments with CIFAR10/100 show that this inference strategy can save up to 44% of the multiply accumulate operations used in inference whilst maintaining model performance, when compared against conventional early-exit methods.

1 Introduction

Over the last decade neural networks have been considered the most powerful method for a number of machine learning tasks such as vision and natural language understanding. This has prompted an increased demand for these methods in commercial and scientific applications. However, these models are resource-intensive and this is compounded when considering embedded applications, where resource availability is limited and may vary at inference time. Hence, to advance the use of neural networks in this domain, consideration must be given to reducing the computational cost in a dynamic manner.

Some progress in model compression has been made through pruning [2, 5, 22] and quantisation [4, 16, 26]. Recent work has manually designed architectures that are optimised for low-cost inference whilst minimising performance drop [7, 8, 11, 13, 19, 28].

However, in rapidly evolving operating environments such as deployed embedded systems, model evaluation has moved beyond accuracy alone. Neural networks may need to dynamically operate in low energy modes whilst still having the ability to match the performance of the original model at peak operating modes. Such operating scenarios will value systems which produce low-cost–lower-confidence outputs that can be improved upon according to operating requirements. We refer to these as *progressive intelligence* systems and the concept of Pareto-optimisation takes a central role in their design, as performance should be maximised at every computational cost point (Figure 1).

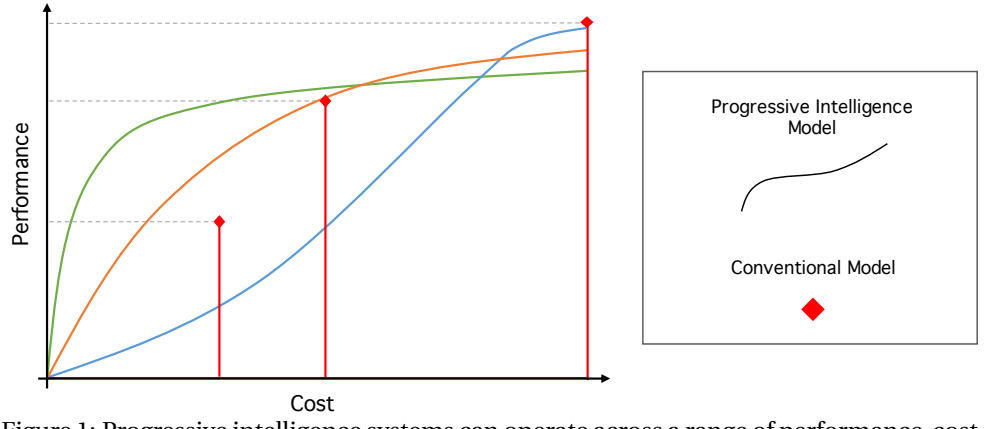

Figure 1: Progressive intelligence systems can operate across a range of performance-cost points whilst valuing low-cost gains in performance, whereas the conventional models operate at a fixed operating point. The conventional models are the least *progressive* due to their fixed nature, followed by the blue progressive intelligence model, as its performance gains are made predominantly at the end of the operating range. The most progressive models make early improvements and build upon them monotonically, like the green and orange models.

Current work in the literature [10, 23, 25] does not consider the complete operating range a dynamic model can operate over. This can be considered from a layer-wise perspective where improvements can be made to internal neural network *representations*. We refer to the total layer-wise activations of a model as its representations. A model's operating range can also be considered in terms of the inference process where performance can be optimised across a continuous range of operating modes, rather than discrete points. The following contributions of our work address these gaps:

- We introduce progressive intelligence and how it can be incorporated into all aspects of the design and optimisation of branched neural networks.
- We present a novel investigation into the effect classification branches have on their backbone using a variety of explainability metrics. This shows how branches improve neural network representations in a layer-wise fashion and how they can be used to make progressive intelligence models.
- We use the scaling of classification confidence thresholds as a way of representing the inference cost of branched neural networks in a continuous manner. We refer to these as *inference modes* and they allow the performance of the model to be understood over its entire operating range.

- We vary backbone scaling and design to understand how they effect the inference modes, allowing us to understand how to best use model design to influence the performance of a branched neural network across its operating range.

- We introduce a new early-exit policy for branched networks resulting in up to 44% MAC operations improvement in inference over conventional approaches, without the need for additional optimisation.

We begin with related work which is covered in Section 2. Then we investigate the effect that classification branches have on intermediate layers of a branched neural network using a variety of metrics and propose an optimal branched network configuration (Section 3). We then present a new analysis of branched network inference modes in Section 4 and investigate the backbone architecture and its interaction with the inference modes. In Section 5, an optimised inference method for branched networks is proposed, which we call entropic mutual agreement. Conclusions are drawn in Section 6.

2 Related Work

Since their inception [22, 23], branched networks have remained an integral part of the dynamic inference research community [9, 10, 24], and more recently the development of optimised inference techniques for them [3, 12].

Work in [21] treats the branches of the network as an ensemble, encouraging diversity between the branches of a language model whilst also preserving their accuracy on the classification task. Work in [25] applies a similar approach to vision models. To introduce information sharing between the classifiers and boost performance they train their early exiting branches as an ensemble, unlike previous works in the vision space. However, this is achieved through additional optimisation. The field has yet to produce an ensemble method in branched network inference that does not require additional optimisation; this is something we address in Section 5.

Previous work in the field fails to consider the entire operating range of a branched neural network and evaluates them at discrete points in the performance-cost space. This is particularly relevant to the progressive intelligence problem as it will allow a complete operating range of a model to be characterised, as in Figure 1. We examine this further in Section 4.

Recent explainability work uses a measure of class separation throughout the network [13], to understand how the classes are distinguished by the model. Centered Kernel Alignment (CKA) has also been proposed to compare network representations by producing pairwise comparisons between the intermediate outputs of every layer of one neural network with that of another. This has been used to analyse different loss functions, model sizes, and architectures. [12, 17, 18].

Finally, intermediate classifiers known as *linear probes* have been proposed as a method of measuring layer-wise class separability [1]. They have also been used in conjunction with some of the above methods in other work [13, 17]. When combined with class separation it can help understand the utility of class separation in classification.

Branched networks are a good starting point for a progressive intelligence system and the methods discussed can allow their inner workings to be probed. These concepts have yet to be combined and the next section will address this.

3 The Effect of the Loss Function

To thoroughly explore the representational impact of branches and gain some insight into how the network behaves when using them we use the three explainability metrics of class separation, linear probe accuracy, and CKA. These metrics are applied to all layers except the final classification layer using a test data set to probe the generalisation of these results.

To measure class separation we use a metric defined in [13] as R^2. The metric takes the ratio of the similarity within a class to that across all classes. Taking one minus this value ensures greater values signify greater class separation and it employs the cosine similarity, according to

$$R^2 = 1 - \frac{\sum_{k=1}^{K}\sum_{m=1}^{N_k}\sum_{n=1}^{N_k}(1-\text{sim}(\boldsymbol{X}_{k,m},\boldsymbol{X}_{k,n}))/(KN_k^2)}{\sum_{j=1}^{K}\sum_{k=1}^{K}\sum_{m=1}^{N_j}\sum_{n=1}^{N_k}(1-\text{sim}(\boldsymbol{X}_{j,m},\boldsymbol{X}_{k,n}))/(KN_jN_k)}. \tag{1}$$

Here, $\boldsymbol{X}$ refers to the activation of an input, n and m refer to a datapoint within the class k and j. N_k and N_j refer to the number of samples within the class and K the total number of classes.

The performance of linear classifiers on the embedding space will reflect the *separability* of the data. We refer to these as *linear probes* [1]. We choose to optimise the networks based on linear probe performance, as this metric is directly related to the classification performance at each exit. Maximising the linear probe performance throughout the network backbone should correspond to improved early exit performance.

We train ResNet18 [6] with four branches positioned equidistantly throughout the model (where the final branch corresponds to the output), according to

$$L_{total}(\hat{y},y;\theta) = \sum_{n}^{N} w_n L_n(\hat{y_{exit}},y;\theta), \tag{2}$$

choosing a range of branch weightings, w_n. A selection of the results are presented in Figure 2.

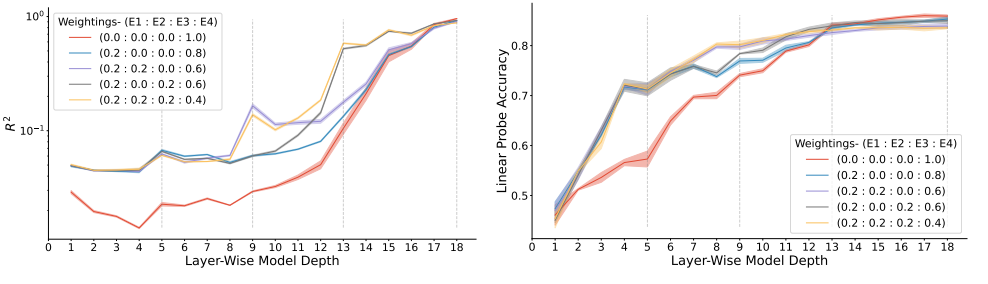

Figure 2: Log of the R^2 metric (left), Linear probe accuracy (right), for different branch weightings of the ResNet18 architecture against layer. The dashed grey lines denote the branch positions in the network and the red line in both figures corresponds to the unbranched ResNet18 architecture. Results were evaluated on CIFAR10 and the width of the lines denote the variance across 3 runs.

The linear probes capture the separability of the classes whereas the R^2 metric captures the magnitude of the actual class separation of the classes. We find that the early branches have a small effect in separating the classes in the representation space, but a large impact in class separability.

Branches positioned later in the network have a greater impact on class separation and the level of class separation at the beginning of the network is increased by the introduction of the first branch. The separation values largely increase monotonically throughout where each branch increases the class separation in the network at its point of connection, but this is followed by a small drop.

The increase in class separation from the early branch corresponds to increased linear probe accuracy. However, the joint optimisation of the final layer prevents the class separation from reaching greater values.

This suggests that restricting class separation early in the model is important for peak classification performance later in the model. However, the linear probe accuracy plot shows that representations created by branched models still allow for greater accuracy earlier in the network. Suggesting that co-optimising the branches allows the class separation to increase without drastically decreasing final layer performance.

To quantify our results we use the area under curve (AUC) of linear probe curves in Figure 2 to give an overall measure of the model's pareto front. For each weight configuration, table 1 presents the AUC and the raw and fractional improvement over the unbranched model.

Table 1: AUC measurements of all branch configurations. Branch weighting is shown in the first column, raw AUC is shown in the second, the third and fourth show raw and fractional improvement over the baseline respectively. The best performing of the single and multiple weighted models are shown in bold.

w - (Exit 1 : Exit 2 : Exit 3 : Final Exit)	AUC	Raw ↑ (%)	Frac. ↑ (%)
Baseline: (0.0 : 0.0 : 0.0 : 1.0)	0.6878	-	-
(0.2 : 0.0 : 0.0 : 0.8)	**0.7276**	**3.98**	**5.78**
(0.4 : 0.0 : 0.0 : 0.6)	0.7266	3.88	5.64
(0.6 : 0.0 : 0.0 : 0.4)	0.7217	3.39	4.93
(0.8 : 0.0 : 0.0 : 0.2)	0.7262	3.84	5.58
(0.0 : 0.2 : 0.0 : 0.8)	0.7096	2.18	3.16
(0.0 : 0.4 : 0.0 : 0.6)	0.7093	2.15	3.13
(0.0 : 0.6 : 0.0 : 0.4)	0.7099	2.21	3.21
(0.0 : 0.8 : 0.0 : 0.2)	0.7078	2.0	2.91
(0.0 : 0.0 : 0.2 : 0.8)	0.6926	0.48	0.69
(0.0 : 0.0 : 0.4 : 0.6)	0.6932	0.54	0.78
(0.0 : 0.0 : 0.6 : 0.4)	0.6919	0.41	0.6
(0.0 : 0.0 : 0.8 : 0.2)	0.6937	0.59	0.86
(0.2 : 0.3 : 0.1 : 0.4)	0.7308	4.3	6.26
(0.2 : 0.2 : 0.2 : 0.4)	**0.7345**	**4.67**	**6.79**
(0.2 : 0.1 : 0.3 : 0.4)	0.7317	4.39	6.39

The results in table 1 suggest that weighting the early branches is most effective when using a single branch. As these branches increase performance early in the network where larger gains can be made. Hence, the area under these curves is improved the most, up to 5.7%. Branches positioned in the third position show little improvement over the baseline model, less than 1%. There is little difference in the AUC measurements for a given branch position, suggesting that branch positioning has a greater effect than weighting when using this metric. When using multiple branches, equally weighting the branches maximises the AUC and thus the overall performance of the model.

CKA has been shown empirically to be maximised between two architecturally identical layers of two neural networks [12]. Thus, we use this to understand the representational similarities between networks trained with branches and those without, allowing us to capture how similarly the networks transform the data. When using linear kernels it can be defined as

$$\text{CKA}(\mathbf{X}_i, \mathbf{Y}_j) = \frac{\text{HSIC}(\mathbf{X}_i\mathbf{X}_i^\top, \mathbf{Y}_j\mathbf{Y}_j^\top)}{\sqrt{\text{HSIC}(\mathbf{X}_i\mathbf{X}_i^\top, \mathbf{X}_i\mathbf{X}_i^\top)}\sqrt{\text{HSIC}(\mathbf{Y}_j\mathbf{Y}_j^\top, \mathbf{Y}_j\mathbf{Y}_j^\top)}}, \tag{3}$$

where $\mathbf{X}_i$, $\mathbf{Y}_j$ are the activations from layers i and j respectively and HSIC refers to the Hilbert-Schmidt-Independence-Criterion [20] which measures the statistical dependence between two distributions; in this work the distributions correspond to the layer representations. Let $\mathbf{K} = k(\mathbf{X}_i, \mathbf{X}_i)$ and $\mathbf{L} = l(\mathbf{Y}_j, \mathbf{Y}_j)$ where k and l are two kernels, the HSIC is defined as

$$\text{HSIC}(\mathbf{K}, \mathbf{L}) = \frac{1}{(n-1)^2}\text{trace}(\mathbf{KHLH}), \tag{4}$$

where $\mathbf{H}$ is the centering matrix defined as $\mathbf{H}_n = \mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, which centers the two matrices in kernel space. We use linear kernels for our analysis and generate a pairwise comparison between the intermediate output of every layer of one neural network and that of another (Figure 3).

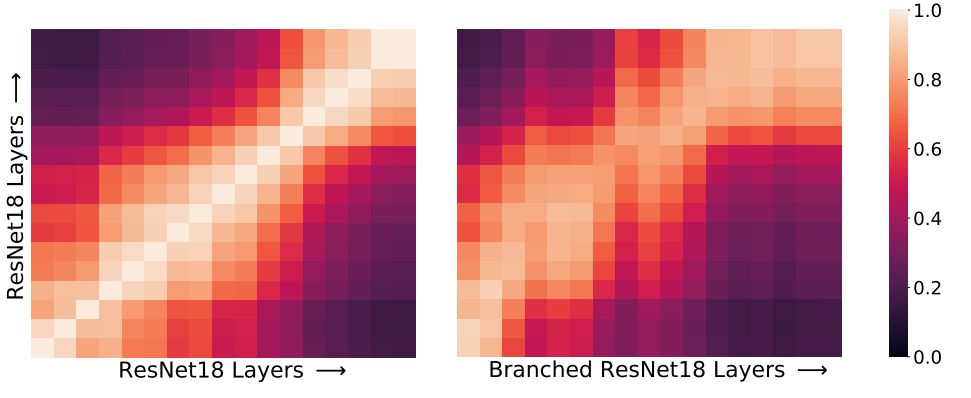

Figure 3: Centred kernel alignment scores between different networks using CIFAR10. Two architecturally identical ResNet18 models (left), a ResNet18 network, and a branched ResNet18 trained with the $(0.2 : 0.2 : 0.2 : 0.4)$ loss weighting configuration (right). The bottom left square on each diagram corresponds to the first layer of each network, and the top right the final layer.

The similarity between the branched and unbranched networks never reaches the maximum values, peaking at ~0.95 in the early layers. Throughout the rest of the network the maximum similarity at a given layer lies between 0.8 and 0.9. In the branch positions there is a clear shift in the similarities of the earlier layers of the branched networks towards the later layers of the unbranched networks. The departure from the diagonal suggests that the branched model is learning similar representations to the unbranched model and that the branches push these earlier in the network.

While improvements to the progressiveness of the networks are evident both qualitatively from Figures 2 and 3, and quantitatively from table 1, it is not evident how this progressiveness transfers to the inference modes, and thus resource savings of such networks. We address this in the next section.

4 The Effect of the Backbone

Early exiting inference uses the intermediate output of the network to determine whether the classification is confident enough. This intermediate output will be produced by the branches which are co-optimised with the backbone. If only the layers preceding the intermediate output are computed, performing an early exit will save the inference cost of subsequent layers for a given input.

An integral part of the early exiting inference is the criteria determining the confidence of classification. The most commonly used metric for confidence is the entropy of the output distribution[10, 23]. The entropy is defined as

$$e(x) = -\sum_{i=1}^{C} p(x_i) \log p(x_i), \tag{5}$$

and an exit is taken if it is less than or equal to a threshold. Here, $p(x_i)$ refers to the probability of the i^{th} class in the output distribution x, and C is the total number of classes. Hence, the entropy is minimised when one class probability approaches one and maximised when all of the class probabilities are equal.

We run inference on the networks and vary the entropy threshold for early exiting, from the maximum possible value of $\log C$, to the lowest possible value, 0. This generates a curve denoting the possible operating points of the network when plotting the average multiply accumulate operations (MACs) usage against accuracy. We refer to these as inference modes. In Figure 4 (top-left) we compare the branched networks trained with different weight configurations.

The highest performing model is that with progressively higher weighted branches. However, the one with the sharpest rise in accuracy is the model with equally weighted branches. Furthermore, the peak accuracy of this network matches the others within its error bounds. Hence, we use this weight configuration for the remainder of our experiments, due to its performance in table 1, in inference, and its simplicity in training setup.

To analyse the effect the backbone has on this distribution we scale the networks. We present MACs vs accuracy plots for ResNet architectures of varied width, depth, and number of branches. Uncertainty is calculated using standard deviations from the mean accuracy value. By assuming the accuracy of the model represents a point of a Gaussian, we can define the confidence interval as $\Delta = \sigma\sqrt{\frac{A(1-A)}{n}}$, where σ represents the number of standard deviations from the mean, n the number of samples, and A the mean accuracy. For the following results, a σ of 3 is used corresponding to a certainty of 99.7%.

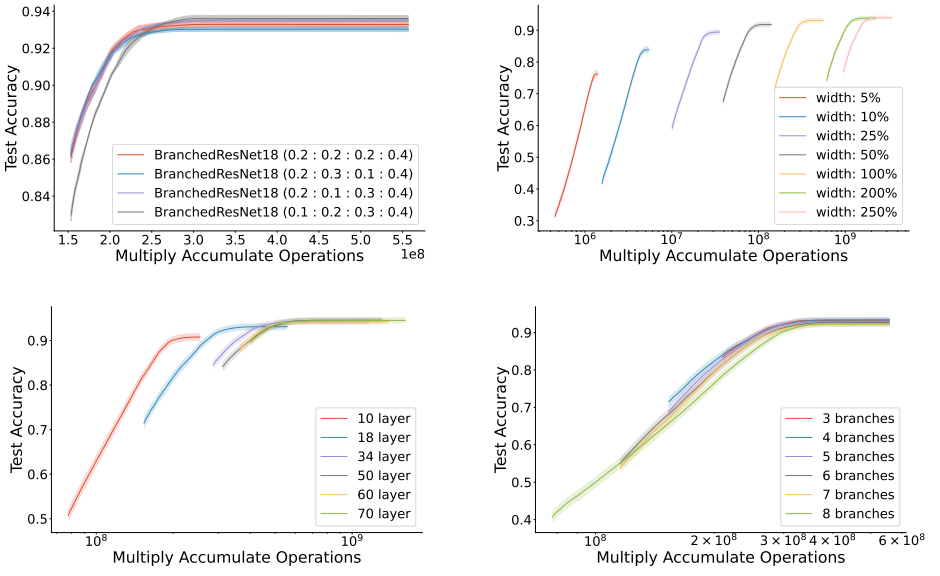

Figure 4: Figures analysing the inference modes of various networks, highlight their operating ranges. The networks trained in Section 3 are compared (top-left). Then the effect of the network width is shown (top-right). Network depth is analysed (bottom-left). The effect of the number of branches is also analysed (bottom-right). Results are taken from CIFAR10. Shaded areas denote a confidence interval of 3σ, in all cases a ResNet18 is used and the architectural component modified is denoted in the legend.

The effect of the width in the ResNet18 architecture is presented in Figure 4 (top-right). It has a consistent effect on the MAC usage of the model, without drastically affecting the classification performance of the model. The performance drop-off starts between a width of 50% and 25%, as the performance of the 50% width model matches the full-size model within the confidence bound. The larger widths do not increase the accuracy outside of the confidence intervals for this dataset. The MACs scale according to the expected ratio $(W_1/W_2)^2$ between two models, where W refers to their respective widths.

The early exiting can at most save ~ 50% of the inference cost (in the far right-hand model), with minimal loss in accuracy, whereas thinning the same model to the standard width has the potential to save ~ 84%. As the model is compressed in width, the benefit of the branching is reduced with the thinner network dropping performance sooner than the wider networks.

The depth of these networks is also analysed, whilst keeping the width consistent with the ResNet18 design. This is shown in Figure 4 (bottom-left). We find that the shallower model has a dramatically reduced minimum accuracy and power range. There are also diminishing returns when using deeper networks. The peak operating powers follow what is expected as there should be a linear relationship between the power used and the depth, but this drops off in deeper models due to the nature of the convolution operation.

The number of equally weighted branches is varied between 3 and 8, shown in Figure 4 (bottom-right). The final branch was given a weight of 0.4 in all cases, the others were hence weighted $0.6/n_{\text{branches}}$. Increasing the number of branches extends the operating

range of the neural network, although at reduced accuracy. The six and seven branched models extend the operating range without significantly compromising the performance at the greater confidence ranges, unlike the 8 branch model. There seems to be no computational benefit of using any less than four branches.

Hence, we have found that width has the greatest potential for changing the operating range of the progressive intelligence network. This can be extended using deeper networks with a larger number of branches.

We also use a more efficient backbone, to understand if these inference mode patterns are backbone agnostic. We use the MobileNet architecture [7, 8, 19]. This is presented in Figure 5 (top-left). We find that when trained using the same hyperparameters, the more efficient architecture is outperformed by the ResNet18, but it operates in a lower power range. We now move our attention to the exit policy of the network to understand how inference modes can be improved for progressive intelligence.

5 The Effect of the Exit Policy

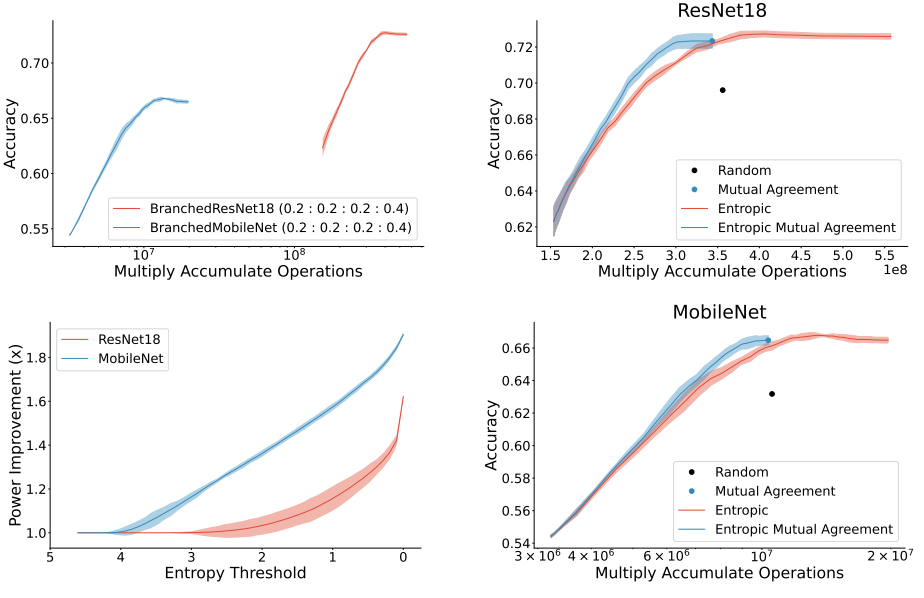

Figure 5: Various figures analysing the inference modes of the entropic mutual agreement exit policy. The MobileNet and ResNet18 architectures are compared (top-left). The mutual agreement exit policy is shown on both the ResNet18 (top-right) and MobileNet (bottom-right). Finally, the power improvement made at a given entropy value is presented for each network (bottom-left). All results are taken from the CIFAR100 test set and shaded areas denote the variance across 3 runs.

We introduce a new exit policy designed to increase the progressiveness of the branched neural network, the *mutual agreement* policy. Most early exiting algorithms in the field fail to consider the branched neural network as a special ensemble and those that do require additional optimisation post-training [25].

Our policy considers two aspects, consistency between branches and confidence at a branch. It exits when a branch is consistent with its previous branch, or if the entropy of the prediction is below a threshold. We call this *entropic mutual agreement* and the exit condition is defined as

$$e_i(x) \leq \alpha \vee \hat{Y}_i(x) \cdot \hat{Y}_{i-1}(x) = 1, \tag{6}$$

where e is the entropy at exit i, α the entropy threshold, and Y a one-hot encoded output.

We present comparisons between these loss functions in both architectures in Figure 5 (top/bottom-right). Figure 5 (bottom-left) shows a comparison between the two loss functions, measuring the power saved by the new exit policy for a given entropy level. From these figures, it is clear the new loss function rises earlier than the entropic policy, and stops sooner, without losing the peak performance of the network. Hence, this makes the network more progressive in offering low-cost inference and more resource efficient at all confidence thresholds. The power improvement peaks at over 1.8× (44% saving) improvement in the MobileNet, and 1.6× (38% saving) in the ResNet18.

6 Conclusions

Progressive intelligence is a framework of machine learning in which the system adapts to produce the best possible outputs according to resource availability. This has a number of applications, namely in dynamic operating environments such as deployed systems on embedded devices, computing servers with variable resource availability, or mobile devices with power-saving modes. A core principle of these systems is pareto-optimality, where multiple objectives are to be maximised.

This work presents branched neural networks as a progressive intelligence system and explores the various ways in which they can be improved and pushed toward pareto-optimality. Various explainability measures are used to understand the effect classification branches have on the inner workings of neural networks when included in the loss function. Linear probe accuracy is used to quantify the progressiveness of intermediate representations in the networks. An optimal branch configuration is identified which maximises the pareto-optimality of the neural network's inner workings.

We present a method of analysing the inference process of a branched neural network that shows the progressive nature of early exiting policies. This allows the operating range of networks to be understood. Using this tool, we analyse several different configurations of a branched ResNet architecture, to understand better how these changes interact with the operating range of early exiting inference. We find changing the depth and width moves the range over which a model can operate. Whilst changing the number of branches extends the range over which the model can operate. Finally, a new early exiting policy is devised, utilising consecutive branch outputs in inference. We find this policy records up to 44% inference cost saving, without the need for additional optimisation.

Future work will investigate new progressive intelligence systems co-designed, and analysed using, the ideas and analysis techniques presented in this work. Such systems will require that they can be modified to produce incremental results, for example, an ensemble system could be configured to operate in such a manner. This work has provided a proof of concept and shown that progressive intelligence can be implemented through the optimisation of branched networks, and should act as a foundation for future implementations of progressive intelligence systems.

Acknowledgements

This work was supported and funded by: The UK Research and Innovation (UKRI) Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1]; the UKRI Turing AI Acceleration Fellowship on Citizen-Centric AI Systems [EP/V022067/1]; the Defence Science and Technology Laboratory (DSTL).

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJ4-rAVtl.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[3] Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, pages 235–244, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3411973. URL https://doi.org/10.1145/3340531.3411973.

[4] Cong Guo, Yuxian Qiu, Jingwen Leng, Xiaotian Gao, Chen Zhang, Yunxin Liu, Fan Yang, Yuhao Zhu, and Minyi Guo. SQuant: On-the-fly data-free quantization via diagonal hessian approximation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JXhROKNZzOc.

[5] Soufiane Hayou, Jean-Francois Ton, Arnaud Doucet, and Yee Whye Teh. Robust pruning at initialization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=vXj_ucZQ4hA.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[9] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3812–3821, 2019.

[10] Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. *arXiv preprint arXiv:2002.10025*, 2020.

[11] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12569–12577, 2019.

[12] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[13] Simon Kornblith, Honglak Lee, Ting Chen, and Mohammad Norouzi. What's in a loss function for image classification? *CoRR*, abs/2010.16402, 2020. URL https://arxiv.org/abs/2010.16402.

[14] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2020.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[16] Xingchao Liu, Mao Ye, Dengyong Zhou, and Qiang Liu. Post-training quantization with multiple points: Mixed precision without mixed precision. *arXiv preprint arXiv:2002.09049*, 2020.

[17] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KJNcAkY8tY4.

[18] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[20] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13 (5), 2012.

[21] Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. Early exiting with ensemble internal classifiers. *arXiv preprint arXiv:2105.13792*, 2021.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[23] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[24] Y. Wang, J. Shen, T. K. Hu, P. Xu, T. Nguyen, R. Baraniuk, Z. Wang, and Y. Lin. Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):623–633, 2020.

[25] Maciej Wolczyk, Bartosz Wójcik, Klaudia Bałazy, Igor T. Podolak, Jacek Tabor, Marek Śmieja, and Tomasz Trzcinski. Zero time waste: Recycling predictions in early exit neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=14-dXLRn4fE.

[26] Kohei Yamamoto. Learnable companding quantization for accurate low-bit neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5029–5038, 2021.

[27] Shunshi Zhang and Bradly C. Stadie. One-shot pruning of recurrent neural networks by jacobian spectrum evaluation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1e9GCNKvH.

[28] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.