Return Curves in Computational Budgets: Analyzing FLOPs for Varying Returns

Andre Mirończuk
andreandrem@proton.me

Abstract

This paper tackles the question of whether FLOPs have varying returns when applied to increasingly demanding tasks, in the active-learning training stage. To test that, a series of experiments was conducted comparing two models across a set of progressively more complex input/output spaces, where the models had the same utility function, architecture, hyperparameters, training pipeline, and differed only in how many total FLOPs they were assigned for training. Results showed that the relative difference between the models' accuracies grew, suggesting that the effectiveness of additional FLOPs increases, as tasks become more demanding, during the active-learning training stage. This trend was consistent between different neural network architectures, FLOP thresholds, compute budget ratios and metrics. The tests were performed in the domain of image recognition across varied datasets. In total, more than 120 models were trained for the experiments.

1 Introduction

This work focuses on understanding how the relative effectiveness of additional FLOPs used during training changes when models are assigned tasks of varying difficulty. Specifically, when the demandingness of the underlying, what we call *True Telos* (tT), changes.

By *True Telos*, we refer to the actual (unknown) "computation" that the network, throughout the optimization process, is gradually impelled to perform during inference. In other words, the final "rules" (optimized heuristics one could say), defined by the search towards them, since those rules are not universe constants — they are a compression of what is needed to predict the future.

For example, nothing explicitly guides chess engines to value a queen more than a pawn; its utility function is completely blind to their existence (it is a part of its *True Telos*, though). Such high-level aspects can be intuitive. This is not the case, however, for the overwhelming majority of others, across domains.

Apart from what the optimization process is actually pushing towards, the aforementioned demandingness of this search can differ. The model architecture, utility function (e.g., loss function), and the algorithm can stay the same, but the underlying task difficulty can vary greatly.

This notion of tasks — or more precisely, *True Teloi* of data & algorithm pairs — having a latent difficulty level can be thought of as a "sororal" version of $P \neq NP$, without any specific time complexities constraints, using a continuous scale — it might be easy to verify that the resulting function works or does not work for a given input, but it can be either easy or hard to converge upon this function (demandingness of *True Teloi* differ).

On a small scale, this work juxtaposes the varying difficulty of *True Teloi* with different training compute budgets to investigate their correlation. We aim to determine whether FLOPs yield varying returns when applied to increasingly demanding tTi. In other words: Does the effectiveness of a FLOP change, when the network is met with tasks of different underlying difficulty, particularly during the active-learning training stage?

To investigate this, we conducted a series of experiments comparing models across a set of progressively more complex input/output spaces. The models shared the same utility function, architecture, hyperparameters, and training pipeline, differing only in their total assigned FLOPs.

We focus on *FLOPs* as they importantly take into consideration both the architecture size and the dataset size. Both of these factors are known to scale, as discussed in Section 2 (Related Work). At larger scales, *petaflop/s-day* (*pfs-day*) could serve as a unit, which consists of performing 10^{15} operations per second for one day, or a total of about 10^{20} operations.

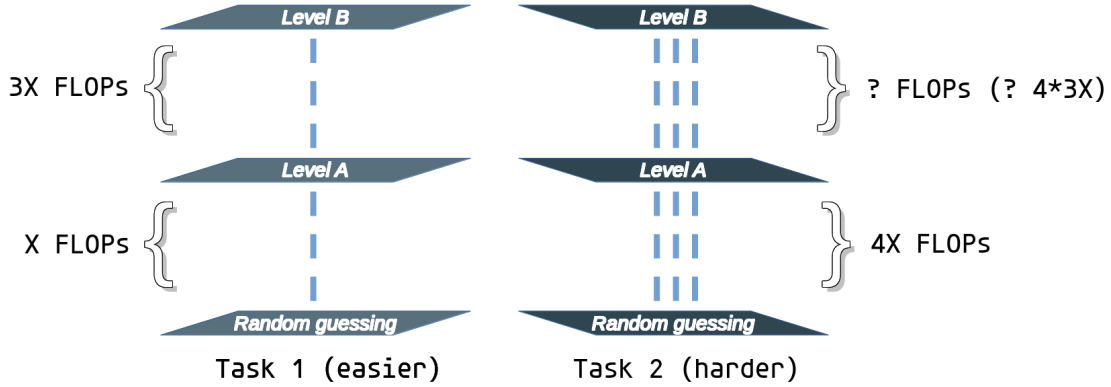

Figure 1: One interpretation of the question tackled by this work. Given two tasks of varying difficulty: If getting to Level A for Task 1 takes X FLOPs, getting from there to Level B takes 3X FLOPs, and for Task 2 to get to Level A it takes 4X FLOPs, then how many FLOPs are needed to get from Level A to Level B in Task 2? Equivalent levels represent comparable relative performance between tasks. Is it less, more, the same, or there is no general rule? In other words: Can the ratio between Random guessing → Level A FLOPs and Level A → Level B FLOPs for Task 1 provide any information about the equivalent ratio for Task 2? What is the general trend?

2 Related Work

The role of computational resources in shaping the performance of machine learning models has been extensively studied, with *scaling curves* being in the forefront of discussions. However, research in this area, including ones that tackled *scaling curves* and *scaling laws*, primarily focused on measuring model performance from a perspective of tasks with statically defined input/output spaces. These studies typically scaled either computation, dataset size, or architectures, rather than examining performance with respect to computation budget for tasks of varying difficulty.

Hestness et al. [1] demonstrated that deep learning systems, empirically exhibit predictable scaling behaviors. Their work focused on power-law relationships in learning curves, showing that generalization error decreases predictably as the size of the available training data grows. However, for example in their image classification experiment only the number of samples per class was scaled, not the number of classes themselves. This approach makes their findings specific

to scenarios where the volume of training samples per class changes while class diversity remains constant, potentially limiting their capability to produce results generalizable across complexity bumps present in ever more demanding real-world tasks — this relates to discussions about so-called emergent behavior and its validity.

Regarding whether testing a particular set of architectures and hyperparameters can reveal universal properties about model training, Hestness et al. [1] showed that model improvements primarily shift the error curve but do not appear to affect the power-law exponent. This suggests a certain universality between different architectures and hyperparameters configurations. While the power-law exponents varied across tested domains, they exhibited consistent trends within each domain.

The authors also noted that the steepness of the learning curve — defined by the power-law exponent — is domain-specific and cannot be universally predicted using theoretical models. This insight directly aligns with the focus of this work, as it emphasizes the need to empirically evaluate computational scaling under varying task complexities.

3 Methodology

First, we will pick a domain and construct a sequence of increasingly demanding tTi. Then, we will pick an applicable model and determine the FLOPs ratio, including the actual FLOP thresholds.

Once these parameters are chosen, we will start the experiment by examining the difference between these models, comparing their performance scores on the first (simplest) environment, that is, the set of input/output spaces. More specifically, we will analyze the ratio between their scores. The score can be different depending on a problem at hand, ranging from final loss on the test set, accuracy to other variously defined metrics, like the total number of legal moves generated, the number of games won, or perplexity.

The reason why the ratio of the model scores is used is that, while the scores for both models could be shown for each environment, it would be visually less intuitive and harder to interpret, since it is not straightforward to gauge relative differences, especially as the scale of the y-axis could differ significantly between different experiments (e.g., different metrics, difference in the base achievable loss), even if the ratio would stay the same.

Next, we repeat the process using the same model architecture and the same number of FLOPs assigned to each model, changing only the environments to more demanding ones, along with their corresponding datasets or training pipelines.

We repeat this process for all the chosen compute budgets across all the constructed datasets or training pipelines. Then we calculate relative differences between models and create graphs displaying ratios of model scores. Our analysis focuses on examining the shape of these curves and understanding their relationship to the meta-hyperparameters chosen. We investigate whether any recurring patterns will emerge across different ratios, architectures and tasks.

3.1 Potential Pitfalls

Importantly, all the FLOP values should be carefully chosen and allow for learning progress on both ends of the learning spectrum for the majority of environments. We are focused no the active-learning training stage. That means that the smaller FLOP value in the chosen ratio needs to allow the network to meaningfully improve (even if relatively little) at each task, escaping from the training stage region dominated by best guessing. Similarly, the higher value should still leave room for improvement for each task, so that we do not approach the irreducible error zone, where model saturates. If either of these conditions occur, the score ratio for the given environment could be considered irrelevant and treated as a baseline.

4 Experiment Design

In order to determine when to stop training the model for a given amount of assigned FLOPs, we will first pre-calculate the amount of the compute needed for one full forward-backward pass, taking into account the batch size. Based on this calculation, the number of forward-backward cycles can be derived. We will keep track of the cycles during training and stop iterating when the limit is hit. If the next forward-backward cycle would cross the threshold, the training is stopped, discarding any remaining computational budget that does not amount to the compute needed for the full batch. This wastes only a marginal number of FLOPs in comparison to the whole run.

For calculating the number of operations performed by the algorithm that runs the model, we will use the in-build PyTorch FlopCounterMode module. As of now it

is an undocumented module. Dedicated third-party profiling libraries performed comparably for this use case.

Because we are dealing with scenarios where computation can be scarce, to minimize the chance of any model spending too big of a portion of available training time in the "assimilating to data" zone dominated by best guessing, we will purposefully set the batch size to be low, so that we perform more frequent updates per FLOP for the same computational budget. In practice, frequent updates using smaller batches, despite their noisier gradients, often contribute to better learning progress compared to infrequent updates with larger, more precise gradients, particularly during the early stages of training. This approach will also be less susceptible to inconveniently shuffled random batches from the training set when working with smaller compute budgets. This aligns with established practices, such as progressive batching strategies, where in the early stages of training smaller batch sizes are used and gradually increased throughout the process.

For similar reasons, there will be no techniques used for preventing overfitting.

4.1 Potential Pitfalls

Measuring FLOPs gives just an estimate that can be imprecise, particularly when faced with non-standard layers. However, these limitations do not impact our analysis, as we are directly only comparing models (algorithms running on certain architectures, if one were to be precise) against themselves for unbiased comparisons between FLOP values. In this scenario, any imprecisions will cancel out.

When it comes to differences between floating-point precision, while for float-16 (F16) and float-32 (F32), the FLOP count remains identical, the actual hardware throughput can be much higher for F16 operations. Additionally there is no distinction between operations executed on tensor cores versus CUDA cores . Although those aspects are important from the practical standpoint (time and energy), they are not directly connected to the relative effectiveness of FLOPs that we are trying to analyze — unless we would be mixing precisions. We will be defaulting to F32 precision for all weights.

With the current implementation, for determining when to stop training the model, a potential issue arises if gradient accumulation is utilized during training. In this case, there could be a scenario where the training is stopped during the accumulation period, without the last optimizer step being performed on accumulated gradients throughout the last set of batches, wasting some of the computation. However, since

we want the batches to be small, and those will fit into the GPU memory without any problem, we will not use accumulating gradients, making it a non-issue.

5 Experiment

Our testing focuses on image recognition. Two variants were constructed for this domain. Tests where performed using a modified training script based on the example from the PyTorch Image Models library [2] (timm). Experiments were performed with both models trained from scratch and pretrained ones. The weights for pretrained models where provided by the timm library.

5.1 Main Variant

The sequence of tasks for the main variant consisted of subsets of Imagenet [3, 4] with progressively more classes added. For the simplest task, the model had to distinguish between 5 classes and had access to 3,250 samples, evenly distributed across all classes — 650 samples each. The validation set consisted of 50 images per class. Each new task introduced 5 more classes. To maintain a consistent total number of passes over the dataset, as a means of maximizing comparability, and to increase the difficulty, the number of samples per class was scaled down for consecutive datasets, so that the total number of samples always remained approximately 3,250. The number of images in the validation set was not scaled and always consisted of 50 samples per class. All images in the dataset were pre-scaled to 160 pixels on the shorter side. The architectures used defaulted to an input size of 224 by 224 pixels, and this default was not changed for this variant. Images were processed accordingly to match this size. Stochastic Gradient Descent served as the optimizer for all models.

Mapping of the number of classes in the dataset to samples per class:

- **5:** 650
- **10:** 325
- **15:** 216
- **20:** 163

- **25:** 130
- **30:** 109

The exact classes used are listed in Appendix A.

Tested architectures:

- Vision Transformer (vit-medium-patch16-clip-224)
- ResNet (resnet18)

ResNet [5] (Residual Network) and Vision Transformer [6] (ViT) architectures were chosen, because they are both well-established, yet considerably different when it comes to their design principles and computational approaches.

ViT adapts the Transformer architecture, leveraging self-attention mechanism instead of convolutions. Unlike convolutional neural networks, ViT operates without an inductive bias, meaning it must learn local and global spatial relationships entirely from data. As a result, Vision Transformer networks generally converge more slowly and require larger datasets than ResNet networks, but they have a higher performance ceiling because the pre-defined spatial assumptions about their inputs are absent.

Vision Transformer

Chosen FLOP thresholds:

- 375 TFLOPs
- 1125 TFLOPs
- 2250 TFLOPs

FLOP ratios:

- 2:1 (2250 : 1125 TFLOPs)
- 3:1 (1125 : 375 TFLOPs)
- 6:1 (2250 : 375 TFLOPs)

ResNet

Chosen FLOP thresholds:

- 150 TFLOPs
- 450 TFLOPs
- 900 TFLOPs

FLOP ratios for the ResNet model remain the same, only the individual FLOP thresholds change.

FLOP ratios:

- 2:1 (900 : 450 TFLOPs)
- 3:1 (450 : 150 TFLOPs)
- 6:1 (900 : 150 TFLOPs)

Reasoning

FLOP values vary between architectures because the experiments were designed with the characteristics of the model architectures and the number of epochs performed over the dataset in mind.

For ResNet and this dataset sequence, 150 teraFLOPs (TFLOPs) correspond to around 4.3 epochs, with one forward-backward pass costing ~ 0.34 TFLOPs (full batch of 32). This means that the total number of full forward-backward cycles for this FLOP threshold equates to 440. To illustrate the difference, 450 TFLOPs for the same architecture yield about 13.3 epochs, while 900 TFLOPs yield around 26.3 epochs.

For similar reasons, and to enable more effective comparison between architectures, the thresholds for the Vision Transformer model were scaled by a factor of 3.5. This adjustment accounts for the architecture's larger size and higher compute requirements per one pass — ~ 0.89 compared to ~ 0.34 teraFLOPs for ResNet. The smallest FLOP value for the Vision Transformer architecture yielded 421 forward-backward passes, corresponding to approximately 4.15 epochs.

5.2 Experimental Variant

In the experimental variant, more contrasting classification tasks within the datasets sequence were attempted, as detailed below. This time, the increase in the number of additional classes was non-linear.

Datasets in Order:

- **Subset of Imagenette:** 5 classes [7]
- **Subset of Imagewoof:** 5 classes (fine-grained classification) [8]
- **Imagenette:** 10 classes
- **Imagewoof:** 10 classes (fine-grained classification)
- **Chars74K EnglishImg GoodImgs:** 62 classes [9]
- **Caltech 101:** 102 classes [10]

The hyperparameters and models were the same as those used in Subsection 5.1 (Main Variant), with the addition of a few larger thresholds used for further testing.

6 Results and Interpretation

6.1 Main Variant

First, we demonstrate how the final graphs are constructed. Figure 2 illustrates the results for the ResNet model architecture, depicting the top-1 accuracy on the validation set for each dataset across FLOP thresholds.

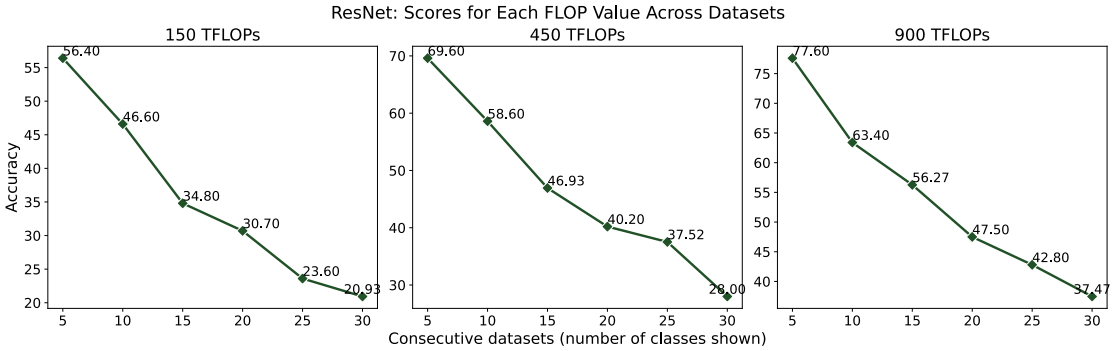

Figure 2: As expected, accuracy declines as tasks become more challenging. The decline remains relatively stable across each FLOP value.

Based on these results, we construct graphs that showcase the ratios between model scores for the chosen FLOP value pairs. Let us being by creating a 6 to 1 ratio graph. To achieve this, we compare the results for 900 and 150 teraFLOPs (Figure 3).

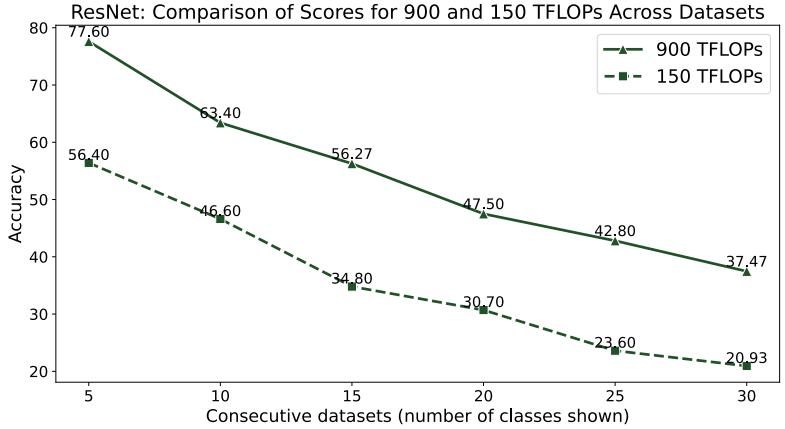

Figure 3: The absolute differences between scores for both runs stay relatively similar, and for this pair it averages approximately 19 percentage points.

Given the performances shown in Figure 3, we can calculate ratios between them, as illustrated in Figure 4.

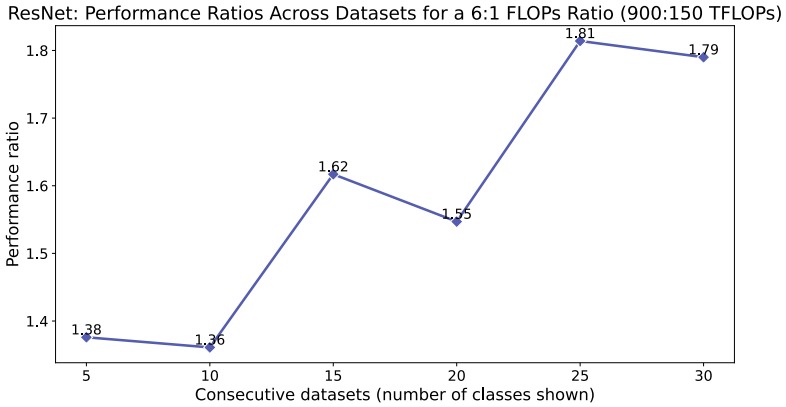

Figure 4: A strong upward tendency is observed. The difference between the relative effectiveness of FLOPs changes. A total of 12 models were trained to generate this graph.

The model with 900 teraFLOPs at its disposal was 38% more effective than the model with 150 teraFLOPs on the simplest dataset. However, as the tasks became more challenging, the relative difference between the models grew. For the hardest task in the sequence, the model with access to more compute — maintaining the same ratio between compute budgets — was 79% more effective. The effectiveness of additional FLOPs appears to increase as tasks become harder for this specific compute budget pair and learning stage.

Figure 4 showed only one juxtaposition. Results for other ratios, architectures, metrics, and pretrained versus untrained models are showcased in the following set of figures (Figures 5–14).

Models trained from scratch

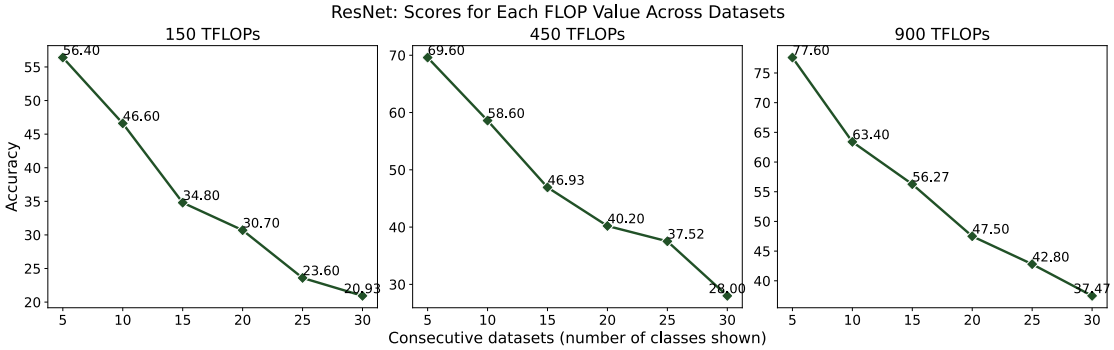

Figure 5: Validation accuracy for the ResNet architecture across consecutive datasets for three different FLOP thresholds: 150, 450, and 900 TFLOPs. The results demonstrate a consistent decline in accuracy as the number of classes increases, with higher compute budgets consistently outperforming lower ones.

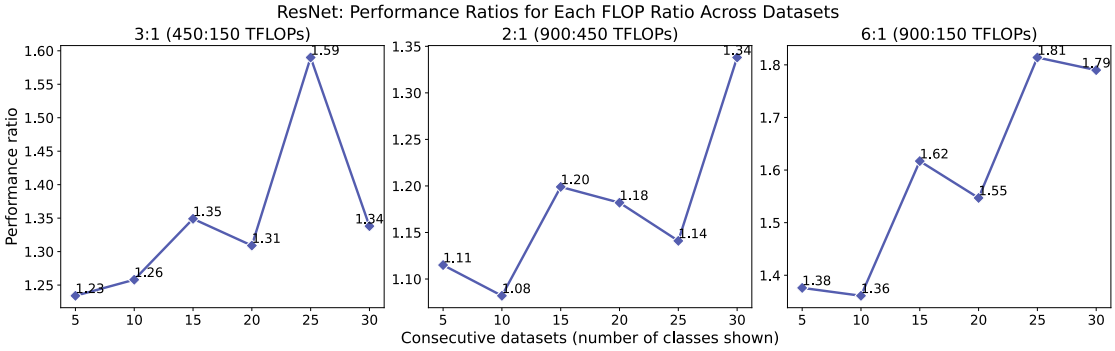

Figure 6: Performance ratios for the ResNet architecture across datasets for three FLOP budget comparisons: 3:1 (450:150 TFLOPs), 2:1 (900:450 TFLOPs), and 6:1 (900:150 TFLOPs). The results highlight a trend where higher ratios amplify performance benefits as tasks become more complex.

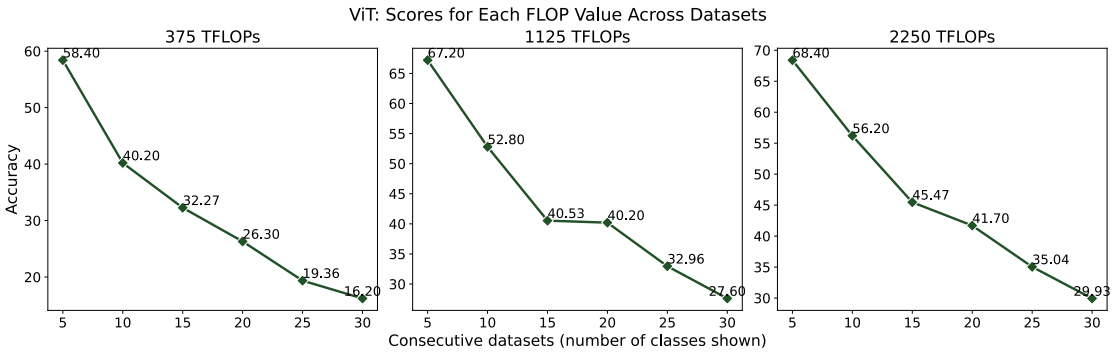

Figure 7: Validation accuracy for the ViT architecture across consecutive datasets for three different FLOP thresholds: 375, 1125, and 2250 TFLOPs. Compared to ResNet, ViT demonstrates a comparable pattern but with consistently lower baseline performance across most tasks.

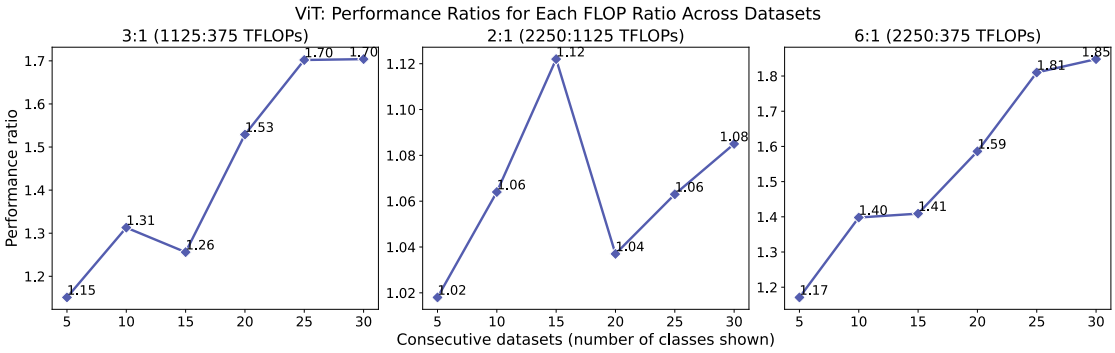

Figure 8: Performance ratios for the ViT architecture across datasets for three FLOP budget comparisons: 3:1 (1125:375 TFLOPs), 2:1 (2250:1125 TFLOPs), and 6:1 (2250:375 TFLOPs). The results once again reveal an increasing performance benefit as task complexity grows, suggesting that additional compute becomes progressively more impactful for more challenging datasets.

Top-5 accuracy

The results for top-5 accuracies remained very similar, as depicted in Figures 9–12. Since there is less variability due to "lucky" or "unlucky" training / testing runs, the results produce smoother lines.

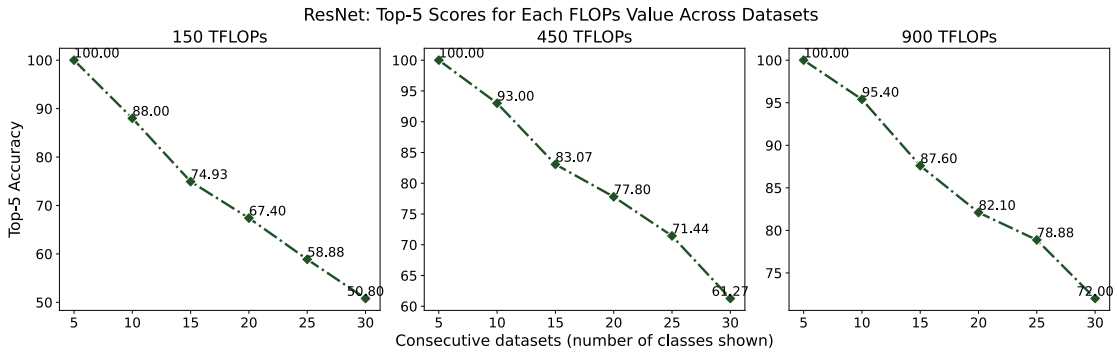

Figure 9: Accuracies for the top-5 metric follow a similar trend.

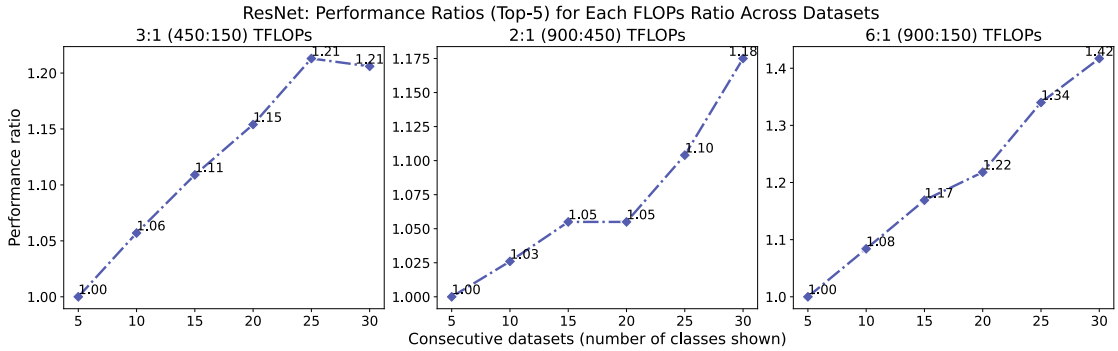

Figure 10: The trajectory for ratios is even more consistent, as top-5 scores introduce less noise.

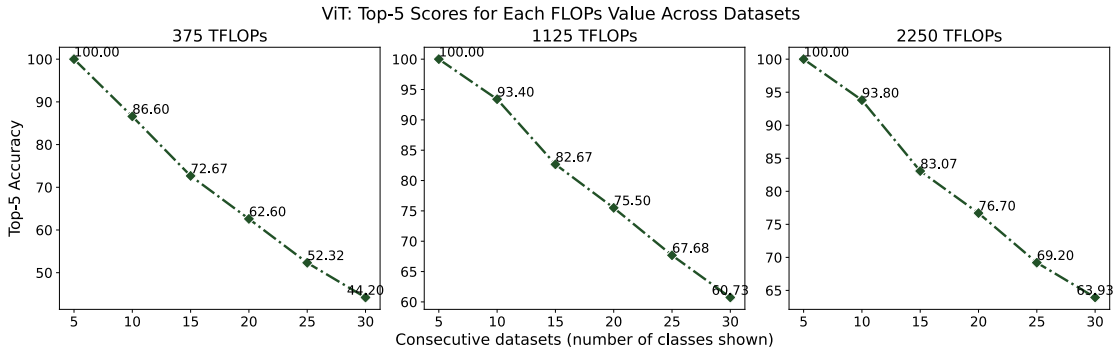

Figure 11: The ViT model closely mirrored the ResNet curves.

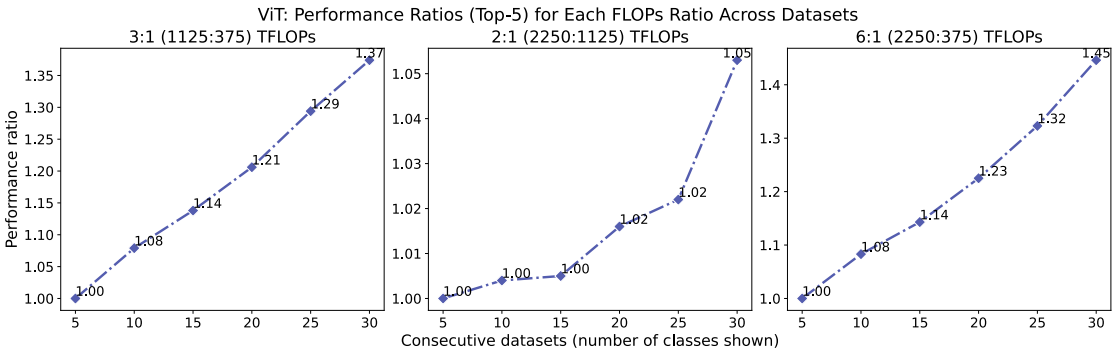

Figure 12: As before, the top-5 metric scores produce smoother graphs.

For all the successfully performed experiments, the performance ratios between different FLOP values gradually increased, while absolute differences between model scores remained relatively constant. Ratios are important, since they show how much better the higher-compute model performs relative to the baseline model. This relative performance difference is crucial because achieving, say, a 10% accuracy improvement is much more impressive when the baseline is 20% compared to when it's 70% — unless the model is approaching its maximum performance limit.

Ratios provide important information that absolute differences alone don't capture, highlighting the relative impact of the performance gap in the context of task difficulty. Together, they give us a more comprehensive picture: while we maintain a similar absolute improvement with more compute, these improvements become relatively more significant as tasks become harder during the active-learning training phase.

Pretrained models

Pretrained models for the ViT architecture generally achieved scores that were too high across FLOP values, including those with the smallest FLOP threshold assigned, which prevented any meaningful comparison. The situation for ResNet was similar, but it consistently achieved lower accuracy compared to ViT, particularly for harder tasks. This resulted in a weaker final comparison, but the trend remained visible, as shown in Figures 13 and 14.

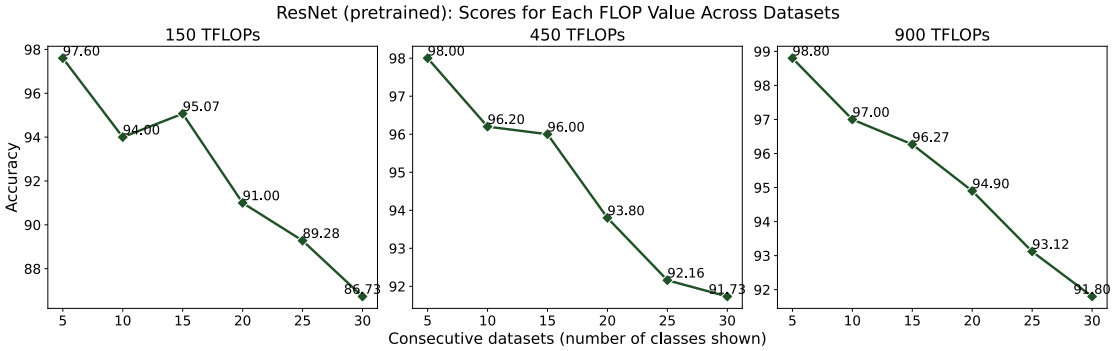

Figure 13: The differences in performance for pretrained ResNet models are small, and the accuracies are high, especially for the first three datasets. This indicates that those tasks were too easy to reveal any meaningful differences.

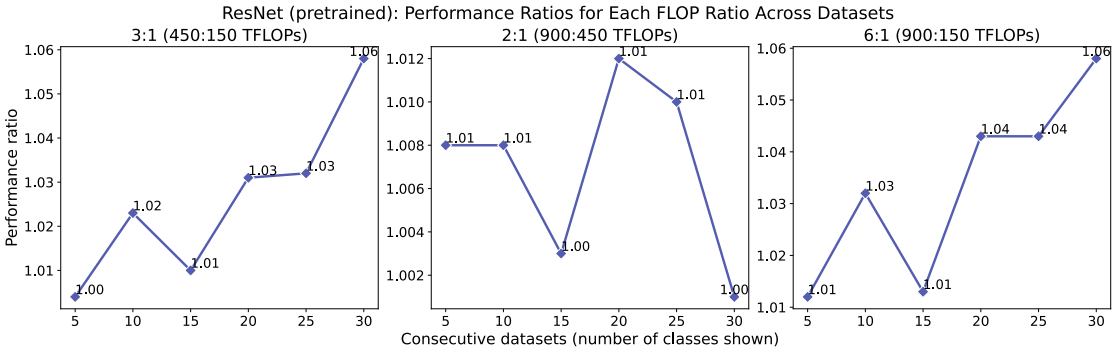

Figure 14: Ratios loosely followed previous trends. With weights initialized the same and "heuristics" already embedded in them, the differences between the models were small, given relatively easy datasets and sparse compute.

Top-5 accuracy

Given that pretrained models were saturating or close to saturation for top-1 accuracy scores, the situation for top-5 accuracy is even more pronounced. This, again, prevented any meaningful comparison on that front.

General results

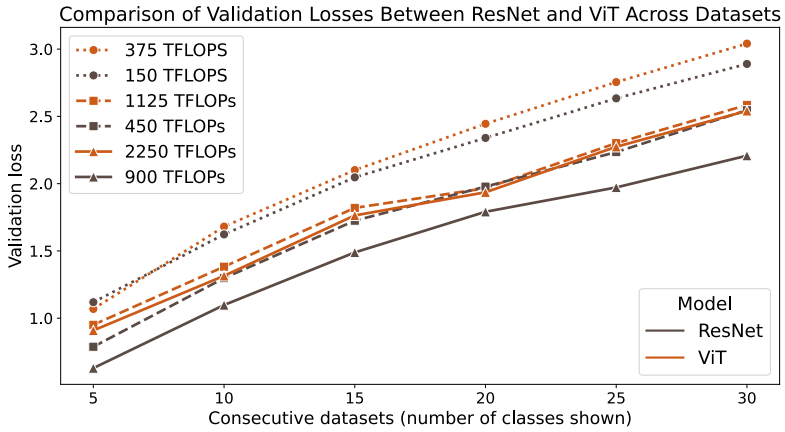

Figure 15: Losses, as expected, follow a predictable trajectory, given the previous results.

In Figure 15, we observe that ViT's validation loss consistently performs slightly worse, particularly for the highest FLOP threshold. For top-1 accuracy, the differences on the most challenging dataset across comparable FLOP values (ResNet–ViT) are as follows: 20.93–16.20, 28.00–27.6, and 37.47–29.93.

This was foreseeable for three reasons. The first is that, although these FLOP values are reasonably analogous, the difference in the number of forward-backward passes between the architectures increases with each increment in FLOP values. The progression is as follows: 19–57–115. For reference, the highest number of forward-backward passes for ResNet was 2,641, while for ViT it was 2,526, resulting in the aforementioned difference of 115 cycles.

This alone would not account for this discrepancy. The second reason is that ViT, having almost twice the number of parameters compared to ResNet (21,675,289 versus 11,189,337), requires more optimizer steps to converge. Smaller models generally converge faster, but they tend to reach saturation more quickly, a consequence of their lower-capacity representations.

Larger models' optimization landscape is more complex, leading to slower convergence initially. As mentioned in Section 5, ViT's forward-backward pass (batch of 32) costs $\sim$ 0.89 teraFLOPs compared to ResNet's $\sim$ 0.34 teraFLOPs. This equates to approximately 2.6 times the computational cost, not $\sim$ 1.93, as with the parameter count difference. This discrepancy stems from the fact that the architec-

ture of ViT (e.g., self-attention mechanism) is computationally more demanding compared to ResNet's convolutional operations, which emphasizes the previously stated assertion that its optimization landscape is more complex.

The last reason was already stated and elaborated upon in Section 5: ViT networks lack an inductive bias, which makes them converge more slowly. They require more data to generalize effectively. However, pretrained ViT models were achieving better results compared to pretrained ResNet models, as mentioned earlier, which aligns with the reasoning presented in Section 5.

6.2 Experimental Variant

When it comes to the experimental variant, the differences between consecutive datasets were much more chaotic due to the types of image classification tasks changing (standard versus fine-grained classification), the sizes of images being inconsistent to a greater extent — leading to them being padded in different ways — and the increase in the number of classes being non-linear.

As a result, with the chosen approach and selected meta-hyperparameters, the experiments turned out to be too messy and, in turn, did not allow for any meaningful comparison between FLOP thresholds. The stark contrast between datasets rendered the compute budgets tested inadequate, as they were poorly prepared to handle all the datasets effectively. Models either saturated too quickly on simpler datasets or failed to progress beyond the best-guessing training stage when faced with more complex ones.

6.3 Conclusions

The results for all the successfully performed experiments consistently demonstrated that the relative effectiveness of additional FLOPs increased as tasks became more challenging, during the active-learning training stage. Across all tested architectures, FLOP thresholds, and compute budget ratios, the models with access to higher compute consistently outperformed their lower-compute counterparts more significantly as task difficulty increased. This trend was most pronounced for the 6:1 and 3:1 compute ratios, and less so, but still consistent, for the 2:1 ratio. This is logical, as the compute difference was the smallest for this ratio, and one could reasonably assume that greater disparities in compute allocation are necessary to observe a stronger performance-gain trend.

However, it is important to emphasize that these observations apply specifically to small compute budgets and the active-learning training stage, meaning the results cannot be extrapolated to scenarios where models approach their performance limits, striving for each percentage point, particularly as the optimization process becomes constrained by the model's inherent capacity rather than its compute budget.

Another critical consideration is that it is not straight forward to design a series of compelling tasks where a considerable amount of training FLOPs can be used, and the network does not saturate for simpler tasks or gets overwhelmed with more complex ones and remains in the best-guessing training zone. That said, these dissimilarities between tasks are an inherent feature of the real world. Bridging the gap between models achieving superhuman or human-expert levels of proficiency on simpler tasks and extending this performance to progressively more complex tTi remains one of the ultimate objectives of the industry.

In conclusion, the findings reinforce the idea that FLOPs are increasingly valuable for harder tasks during the active-learning phase. However, they also highlight the importance of designing experiments and benchmarks that reflect the existence of complexity bumps present in real-world problems.

7 Future Work

This work opens several directions for future investigation. First, while the experiments focused on image recognition and dealt with small compute budgets, extending this analysis to other domains with the inclusion of much higher FLOPs accessible to the models, would help validate the generality of the findings about FLOP returns. Domains involving autoregressive transformers [11], utilized for statistical modeling of token streams, and reinforcement learning domains could offer particularly interesting test cases, as they often present different computational challenges and utility landscapes. Some of those systems also managed to scale to superhuman or human-expert levels in certain tasks and the effectiveness of FLOPs in this context was the primary motivation for pursuing work in this area, and in this aspect of scaling.

Beyond domain expansion, investigating a broader spectrum of tasks within each domain would be valuable. For instance, within computer vision, examining how FLOP returns vary across object detection, semantic segmentation, and instance segmentation could reveal task-specific patterns in computational efficiency.

A particularly promising direction involves testing these findings in the context of deep learning guided program synthesis. The Karel domain, as demonstrated in Jin and Rinard (2024) [12], could serve as a useful testing ground for examining how FLOP returns scale when the output space involves structured programs, where the model needs to handle maintaining increasingly complex representations of the program semantics in its weights throughout the autoregressive program generation. The action space of the agent would be progressively broadened. This could shed light on whether the observed patterns of increasing returns hold in scenarios where the model must learn to generate syntactically valid programs.

Additionally, applying this analysis framework to advanced architectures like MuZero [13] could provide insights into how FLOP returns scale in models that combine planning and learning. This could be particularly relevant for understanding computational efficiency in hybrid systems that must balance multiple types of computation. MuZero, in particular, showed impressive performance across Atari games, Go, Chess, and Shogi making it a prime candidate for studying computational scaling patterns that generalize.

With access to greater computational resources, future work could explore these relationships at larger scales. This would involve:

- Testing with more sophisticated model architectures
- Examining larger datasets with higher complexity
- Investigating broader ranges of FLOP allocations
- Analyzing longer training trajectories

References

[1] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017. URL `http://arxiv.org/abs/1712.00409`.

[2] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. Accessed: 2024-11-30.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[4] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 776–794. Springer, 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL `https://arxiv.org/abs/2010.11929`.

[7] Jeremy Howard. Imagenette. https://github.com/fastai/imagenette, 2019. Accessed: 2023-12-19.

[8] Jeremy Howard. Imagewoof. https://github.com/fastai/imagenette, 2019. Accessed: 2023-12-19.

[9] Teófilo Emídio de Campos, Bodla Rakesh Babu, and Manik Varma. Character recognition in natural images. In *International Conference on Computer Vision Theory and Applications*, 2009. URL `https://api.semanticscholar.org/CorpusID:4826173`.

[10] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[12] Charles Jin and Martin Rinard. Emergent representations of program semantics in language models trained on programs, 2024. URL `https://arxiv.org/abs/2305.11169`.

[13] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL `http://arxiv.org/abs/1911.08265`.

A Sequence of Datasets: Additional Details

Below are the classes included in each dataset for the sequence constructed around Imagenet dataset (number of classes on the left):

- **5:** cocktail shaker; throne; park bench; obelisk; borzoi, Russian wolfhound
- **10:** Previous classes plus green mamba; jean, blue jean, denim; theater curtain, theatre curtain; pirate, pirate ship; goose
- **15:** Previous classes plus vacuum, vacuum cleaner; carbonara; car wheel; red fox, Vulpes vulpes; boxer
- **20:** Previous classes plus honeycomb; tile roof; lorikeet; football helmet; Dutch oven
- **25:** Previous classes plus window screen; computer keyboard, keypad; Rottweiler; mousetrap; pickup, pickup truck
- **30:** Previous classes plus boathouse; milk can; safety pin; English foxhound; wild boar, boar, Sus scrofa

Subsets for Imagenette and Imagewoof look as follows:

- **Imagenette:** church; parachute; golf ball; English springer; French horn
- **Imagewoof:** golden retriever; Rhodesian ridgeback; Australian terrier; Old English sheepdog, bobtail; dingo, warrigal, warragal, Canis dingo