Auto-Compressing Networks

Vaggelis Dorovatas[1,2]
vdorovatas@hotmail.gr

Georgios Paraskevopoulos[3]
g.paraskevopoulos@athenarc.gr

Alexandros Potamianos[1,2]
potam@central.ntua.gr

[1]National Technical University of Athens
[2]Archimedes RU, Athena RC
[3]Institute of Language and Speech Processing, Athena RC

Abstract

Deep neural networks with short residual connections have demonstrated remarkable success across domains, but increasing depth often introduces computational redundancy without corresponding improvements in representation quality. We introduce Auto-Compressing Networks (ACNs), an architectural variant where additive long feedforward connections from each layer to the output replace traditional short residual connections. By analyzing the distinct dynamics induced by this modification, we reveal a unique property we coin as *auto-compression*—the ability of a network to organically compress information during training with gradient descent, through architectural design alone. Through auto-compression, information is dynamically "pushed" into early layers during training, enhancing their representational quality and revealing potential redundancy in deeper ones. We theoretically show that this property emerges from layer-wise training patterns present in ACNs, where layers are dynamically utilized during training based on task requirements. We also find that ACNs exhibit enhanced noise robustness compared to residual networks, superior performance in low-data settings, improved transfer learning capabilities, and mitigate catastrophic forgetting suggesting that they learn representations that generalize better despite using fewer parameters. Our results demonstrate up to 18% reduction in catastrophic forgetting and 30-80% architectural compression while maintaining accuracy across vision transformers, MLP-mixers, and BERT architectures. These findings establish ACNs as a practical approach to developing efficient neural architectures that automatically adapt their computational footprint to task complexity, while learning robust representations suitable for noisy real-world tasks and continual learning scenarios.

1 Introduction

Deep learning has achieved significant breakthroughs across diverse tasks and domains [28, 30, 6]; however, it still lacks the flexibility, robustness, and efficiency of biological networks. Modern models rely on deep architectures with billions of parameters, leading to high computational, storage, and energy costs. Architecturally, these large models are primarily characterized by short residual connections [20], a design initially developed to enable robust training of deep neural networks via backpropagation by providing stable gradient flow through these exceptionally deep networks. These skip connections establish a network topology where multiple information pathways are created [53], resulting in an ensemble-like behavior that delivers more efficient training and superior generalization compared to traditional feedforward networks.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Historically, since the emergence of Highway Networks [47], which first proposed additive skip connections researchers have explored numerous architectural variations. Residual Networks (ResNets) [20] removed learned gating functions and adopted direct identity skip connections becoming the industry standard. DenseNets [22] utilized feature concatenation instead of addition, while FractalNets [29] introduced a recursive tree-like architecture combining subnetworks of multiple depths to further enrich feature fusion. More recent works include learned weighted averaging across layer outputs [37], application of attention mechanisms across block outputs [14] and denser connectivity patterns between network nodes [63]. Other works have explored adding scalars to either the residual or block stream to improve performance, training stability and representation learning [43, 3, 62, 16]. In neural machine translation, researchers have drawn inspiration from both vision and language domains to combine information from different layers, enabling richer semantic and spatial propagation throughout the network [11, 59].

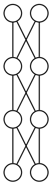
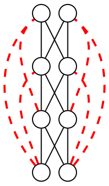
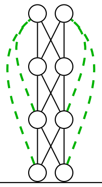
Arch	Connectivity	Forward Propagation	Backward (Gradient) Propagation
FFN		$y_F = \prod_{i=1}^{L} w_i x_0$	$\frac{\partial y_F}{\partial w_i} = \underbrace{\left(\prod_{k=i+1}^{L} w_k\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1} w_m\right)}_{\textit{forward}\text{ term}} x_0$
ResNet		$y_R = \prod_{i=1}^{L}(1 + w_i)x_0$	$\frac{\partial y_R}{\partial w_i} = \underbrace{\left(\prod_{k=i+1}^{L} (1 + w_k)\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1} (1 + w_m)\right)}_{\textit{forward}\text{ term}} x_0$
ACN		$y_A = \left(1 + \sum_{i=1}^{L} \prod_{j=1}^{i} w_j\right) x_0$	$\frac{\partial y_A}{\partial w_i} = \underbrace{\left(1 + \sum_{j=i+1}^{L} \prod_{k=i+1}^{j} w_k\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1} w_m\right)}_{\textit{forward}\text{ term}} x_0$

Table 1: Connectivity (2D case), Forward and Backward Propagation (1D linear case) for FFN, ResNet, and ACN architectures.

While Residual Networks, as discussed, offer efficient and effective training especially in very deep architectures, there are various works that explore behaviors of these architectures that may harm generalization. In [53, 48] it is shown that these architectures exhibit a notable resilience to layer dropping and permutation, which could indicate potential redundancy of some layers (i.e. removing a layer does not affect the network). In [23], it was further observed that dropping subsets of layers during training can reduce overfitting and improve generalization. In a related study [1], the authors showed that introducing skip connections between layers can lead to parts of the network being effectively bypassed and under-trained. In [62], they show that unscaled residual connections can harm generalization capabilities in generative representation learning. Finally, more recently, research has revealed substantial parameter redundancy and inefficient parameter usage in large-scale foundation models, particularly within their deeper layers [18, 7]. All these observations can be unified under the perspective that, although residual architectures facilitate training via multiple signal pathways, these same pathways can sometimes act as shortcuts that cause certain components to be either underutilized or prone to overfitting—ultimately limiting effective generalization.

Inspired by these observations, in this work we explore whether we can design alternative architectures that preserve the key advantages of Residual Networks—such as multiple signal pathways and efficient gradient flow—while addressing issues like redundancy and shortcut overuse, ultimately enabling more efficient parameter utilization and improved representation learning. To this end, we propose an architectural variant where additive long feedforward connections from each layer to the output replace traditional short residual connections as shown in Table 1, introducing Auto-Compressing Networks (ACNs). ACNs showcase a unique property we coin as *auto-compression*—the ability of a network to organically compress information during training with gradient descent, through architectural design alone, dynamically pushing information to bottom layers, enhancing their representational quality, and naturally revealing redundant in deeper layers. We theoretically investigate the emergence of this property by analyzing the gradient dynamics of networks with different connectivity patterns. As illustrated in Figure 1, ACNs demonstrate layer-wise training patterns in which early layers receive significantly stronger gradients during the initial stages of training, in contrast to the more uniform gradient distribution observed in Residual Networks. Next, we empirically demonstrate a broad range of advantages that ACN-learned representations offer compared to residual or feedforward architectures, including: enhanced information compression, superior generalization, reduced catastrophic forgetting, and efficient transferability. Our contributions can be summarized as:

- We introduce Auto-Compressing Networks (ACNs), an architecture that organically compresses information into a subset of the full network's layers, through architectural design alone.
- We provide a detailed analysis of the gradient dynamics of ACNs, along with residual and feedforward networks, shedding light on their distinct behaviors and arguing that different connectivity patterns result in unique training regimes that drive distinct learned representations.
- We implement ACNs in fully connected and transformer-based architectures, finding that they match or outperform residual baselines, while 30–80% of top layers effectively become redundant as information concentrates in the lower layers. Notably, ACNs are hardware-friendly and require no specialized software.
- We show that ACNs learn representations that are more robust against noise and generalize better in low-data regimes compared to residual architectures.
- ACNs reduce catastrophic forgetting by up to 18% compared to residual networks in continual learning by preserving capacity for new unseen tasks.
- ACNs outperform regularization-based approaches at transfer learning without requiring hyperparameter tuning.

2 Auto-Compressing Networks

In ACNs [1], residual short connections are replaced with long feedforward connections, as described in Eq. 1 for a network of depth L, and shown [2] in Table 1:

$$x_i = f_i(x_{i-1}), \qquad y = \sum_{i=0}^{L} x_i \tag{1}$$

In ACNs the output of each layer is directly connected to the output of the network, and thus is directly optimized by the objective function during gradient descent training. Furthermore, the number of possible shortcuts is equal to the number of layers, L. We find this simplification maintains the improved signal flow that shortcut connections provide, while also introducing the ability to detect potential parameter redundancy in the architecture[3]. We note that ACNs differ structurally from other models employing long connections, such as DenseNets [22] and DenseFormer [37], which are residual networks variants, as previously discussed. These models connect each layer to all preceding layers whereas ACNs connect each layer only to the output, leading to a distinct structural design. An overview of these models along with an extended literature review is provided in Appendix A. For completeness, we also provide a direct, fair comparison against other residual variants, such as DenseFormers and DenseNets, in Appendix E.

[1] Code for the paper is available here.

[2] The connections shown for ResNet (red) are implicit, resulting from the residual summation. A detailed example is provided in Appendix D.

[3] Note that long connections are a strict subset of the 2^L shortcut connections in residual networks.

2.1 Gradient Propagation Across Network Architectures

In this section, we examine and compare the forward and backward pass (gradient flow) dynamics of three architectures: traditional feedforward networks (FFN), residual networks (ResNet), and the proposed auto-compressing networks (ACN), based on the equations of Table 1. See Appendix C for a detailed derivation of the gradient equations for 1D linear neural networks.

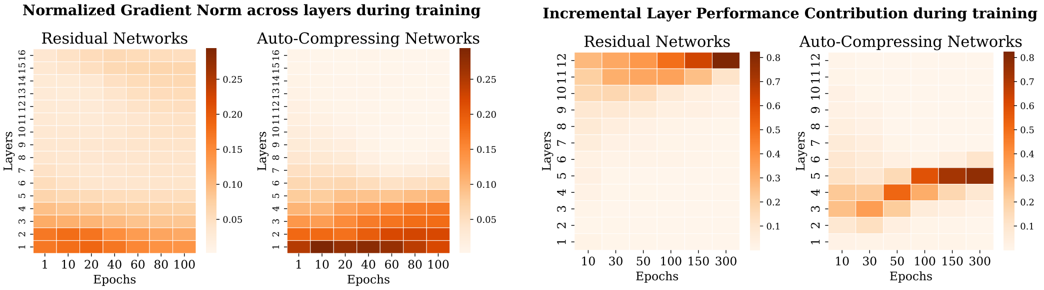


Figure 1: **(left)** ACNs vs Residual Networks gradient flow across layers during training for MLP-Mixer architecture [50] on CIFAR-10 [27], showcasing implicit layer-wise training and information concentration on the bottom layers for ACNs. **(right)** ACNs vs Residual Networks incremental performance contribution across layers for ViT architecture [10] on ImageNet-1K, revealing auto-compression by gradual layer-wise training in ACNs.

2.2 Emergent Gradient Paths

The *forward* and *backward* components: As shown in Table 1, each gradient w_i (see $\frac{\partial y_*}{\partial w_i}$ in the 3rd column of the respective table) decomposes into forward and backward terms. The forward term determines gradient and forward propagation stability (whether the signal vanishes or explodes), while the backward term influences learning. For backward paths, 1D FFNs contain a single path, while 1D ACNs have $L - i + 1$: one direct path using the layer's own long connection to the output, plus $L - i$ additional paths where gradient flows from each subsequent layer's long connection and back through the network. 1D ResNets have 2^{L-i} paths since at each layer there are two options: flow through the network or follow the residual connection. It is also worth observing that ACNs feature a forward term identical to FFNs for intermediate layers (single path), while their backward components is closer to ResNets since it consists of multiple paths.

The Backward component (Full Gradient - **FG**) can be further decomposed into a *Network-mediated Gradient* (**NG**) component that is scaled by network weights and backpropagates information through (a subset of) the network and a *Direct Gradient* (**DG**) component that directly connects from the output to each layer, shown as the term "1" in the backpropagation equations of ACNs and ResNets in Table 1[4] This direct path acts as an information super highway, especially early in training where weights are typically initialized close to zero, informing each layer directly how to contribute towards lowering the optimization objective. Finally, the **DG** contribution is more significant for ACNs compared to ResNets, due to ACNs' linear (rather than exponential) total gradient path count.

Unlike the symmetric forward and backward terms of ResNets and FFNs, ACNs gradients, as argued, consist of a single forward path and multiple backward paths. This design creates an implicit layer-wise training dynamic, where deeper layers are trained at a slower rate compared to earlier layers, since they have a **weaker forward component** (assuming close-to-zero initialization) and a **smaller number of backward paths**. Further, when compared to ResNets, ACNs have a stronger contribution during backpropagations from the **DG** path (vs. **NG**) and this effect becomes more pronounced for deeper networks and for the early layers. For example, when training the second layer of a 1D $L = 12$ layer network, **DG** is one of 11 ACN backward paths, while for ResNets the **DG** is competing with another 127 paths (of the **NG** term). This further accelerates training of the early layers.

Thus, we postulate that: 1) a strong **DG** component coupled with a weaker feed-forward signal leads implicitly to efficient layer-wise training, and 2) architecturally-induced layer-wise training results

[4] This backward path has been previously explored as an alternative to traditional backpropagation and is typically referred to as Direct Feedback Alignment (DFA) in the literature [35, 40].

inadvertently in a form of **structural learning** where information is naturally pushed to early layers, i.e., later layers will become redundant (effectively identity mappings) if the earlier layers can already solve for the task. We refer to this new class of networks as **auto-compressors** since they naturally "shed" their redundant layers during backpropagation simply via architectural design. These claims are experimentally validated in the rest of the paper.

3 ACNs in Practice: Information Compression and Gradient Flow

Next, we implement auto-compressing networks on top of state-of-the-art neural architectures across diverse tasks and datasets. We implement ACNs using variants of the Transformer [52] for language and vision tasks and MLP-Mixer [50] for vision tasks. This allows us to evaluate our approach on diverse benchmarks including image classification (CIFAR-10 [27], ImageNet-1K [41]), sentiment analysis, and language understanding (BERT [9] on GLUE [54]).

In ACNs, for each input token, the final output vector y^t is the sum of all intermediate layer representations plus the input embedding (Eq. 1). For classification, we apply pooling for images or use the `[CLS]` token for text. For a network of depth L, predictions using k intermediate layers compute y_k^t (sum of representations up to layer k) for each token, passed to a single shared classification head. This yields L+1 sub-networks, from input-only to full-network. For residual network baselines, at depth k, we take the output y_k^t of layer k as our $k+1$ subnetwork's output.

Our experiments begin by empirically validating the main claim established in the previous section, i.e., the presence of a strong **DG** component coupled with implicit layer-wise training dynamics drives auto-compression, a property that resembles a form of structural (layer-wise) learning.

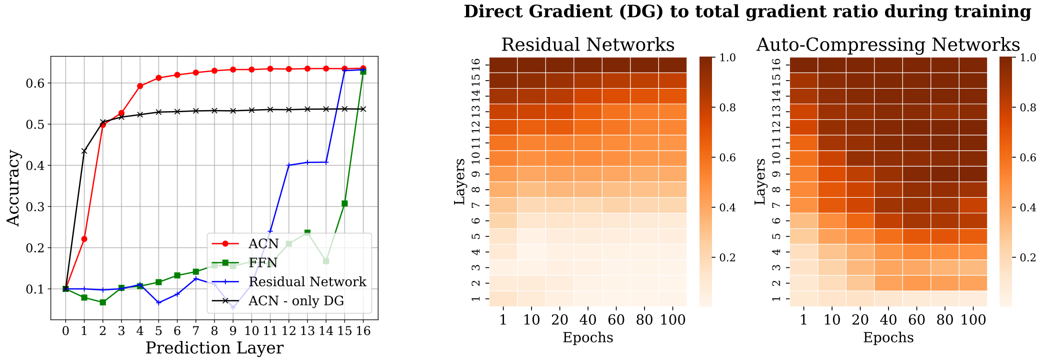


Figure 2: Results for the CIFAR-10 task using the MLM-Mixer base architecture: **(left)** ACN variants achieve auto-compression needing only a few layers to achieve good performance. **(right)** The ratio of direct gradient **DG** to the total gradient **FG** is significantly higher in early layers for ACNs.

To this end, We train feedfoward (FFN), residual and auto-compressing variants incorporated in the MLP-Mixer architecture on CIFAR-10 dataset for 100 epochs. To emphasize the role of the DG gradient in auto-compression, we also train an ACN variant receiving gradients only from the long connections (ACN - only DG component). In Figure 2(left), we show classification accuracy plotted against network depth (layer probing) and observe that among ACNs, FFNs, and Residual Networks, only ACNs exhibit auto-compression. Moreover, ACNs utilizing only the direct gradient (**DG**) still achieve significant auto-compression, highlighting the importance of a strong **DG** component to achieve this behavior[5] and explaining why FFNs do not exhibit auto-compression, as they lack a direct gradient term (Equation 4). In the case of Residual Networks, we previously argued that the exponential number of gradient paths substantially diminishes the influence of the direct gradient (**DG**)

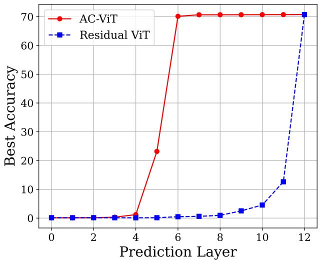


Figure 3: Performance of intermediate layers of AC vs Residual Vision Transformers trained on Imagenet-1K.

[5] ACNs with only the **DG** component under-perform, underpinning the importance of the **NG** component for maximizing performance.

on the overall gradient, a component crucial for auto-compression. To further illustrate this, Figure 2(right) presents the ratio of **DG** to the full gradient **FG** across layers during training for both AC and Residual variants. The results indicate a significantly higher **DG** to **FG** ratio in ACNs, confirming the increased contribution of direct gradients in the early layers of auto-compressing architectures compared to residual networks and explaining the auto-compression property. Furthermore, from Figure 1 we observe that ACNs demonstrate a concentrated gradient pattern with stronger signals in early layers and stronger patterns of **layer-wise learning**. Residual Networks exhibit a more "uniform layer learning" pattern, whereas deeper layers show increasing gradient contribution in later epochs, suggesting task-specific adaptation as training progresses. Interestingly, the pattern observed in Residual Networks indicates that high gradient norms are primarily concentrated in the early and deep layers, while middle layers receive significantly lower gradients, suggesting potential redundancy.

4 ACNs *compress* more

4.1 Auto-Compressing Vision Transformers and MLP-Mixers

Next, we evaluate ACNs in the context of transformer architectures by implementing an auto-compressing variant of Vision Transformer (ViT) [10]. We train a Vision Transformer (ViT) with long connections (AC-ViT) from scratch on the ILSVRC-2012 ImageNet-1K, following the training setup in the original paper. For both models we use 256 batch size due to memory constraints. AC-ViT converges at 700 epochs, while the Residual ViT converges at 300 epochs. As shown in Fig. 3, AC-ViT reaches top performance at only 6 layers while the vanilla ViT needs all 12 layers to reach similar performance, effectively suggesting that *ACNs can improve inference time and memory consumption without sacrificing performance*. To gain more intuition about the training dynamics and task learning of the two variants, in Figure 1(right) we plot the incremental layer performance contribution (difference in accuracy of subnetwork $i + 1$ to subnetwork i) to track the behavior of intermediate layers throughout training. The key observation is that ACNs are trained in a layer-wise fashion, while the residual variant performs task-learning in the last 2-3 layers, effectively utilizing the full network to achieve top performance.

The effect of *Task Difficulty* Intuitively, overparameterized networks trained on easier tasks should demonstrate higher levels of redundancy. Therefore, ACNs should converge to utilizing fewer layers as task difficulty decreases. To verify this, we use the number of classes as a proxy for task difficulty for image classification on the CIFAR-10 dataset [27]. Specifically, we create subsets of 2, 5, and 10 classes, the assumption being that binary classification should be easier than 10-class classification. For this experiment we utilize MLP-Mixer [50] and train two variants, the original MLP-Mixer with residual connections and the modified MLP-Mixer with long connections (AC-Mixer). Results are presented in Fig 4. We observe that indeed *AC-Mixer converges to solutions with larger effective depth, as the task "difficulty" increases*. Specifically, in this experiment, ACN needs 8, 10 and 12 layers for the 2, 5 and 10-class classification problem, respectively. In contrast, the Residual Mixer converges to solutions where the full depth of the network is utilized, irrespective of the task difficulty [6].

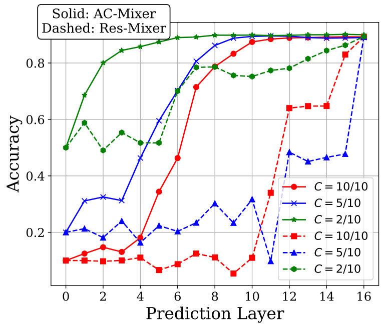


Figure 4: Performance of the intermediate layers as the number of classes (and examples) in the CIFAR-10 dataset increases from 2, to 5 to 10 classes: (a) Residual Mixer vs (b) AC-mixer (C denotes the number of classes in the subset).

4.2 Auto-Compressing Encoder Architectures for Language Modeling

In this section, we conduct a preliminary study on the effectiveness of the ACN architecture in general pre-training (masked language modeling with a BERT architecture) followed by fine-tuning. The

[6] The Residual Mixer was trained for 300 epochs, while AC-Mixer for 420 epochs to reach the performance of its residual counterpart.

results show that ACNs learn compact representations that: 1) achieve on-par performance with the residual architecture on transfer learning tasks, while utilizing significantly fewer parameters, and 2) complement post-training pruning techniques, enhancing their effectiveness.

4.2.1 Masked Language Modeling and Transfer Learning with ACNs

We compare the ACN and residual architectures in the standard BERT pre-training and fine-tuning paradigm. Using the original BERT pretraining corpus (BooksCorpus [64] and English Wikipedia), we train both architectures to equivalent loss values; the AC-BERT variant requires two epochs vs one epoch for the residual baseline. Following pre-training, we fine-tune both models on three GLUE benchmark datasets [54]: SST-2 sentiment analysis [45], QQP paraphrasing, and QNLI question answering [38]. Figure 5(left) demonstrates a key advantage of the ACN architecture: it naturally converges to using significantly fewer layers (approximately 75% less layers) while maintaining performance comparable to the full residual network. These results suggest promising applications for ACNs in large language models, where pre-training could be performed with long connections, allowing downstream tasks to adaptively utilize only the necessary subset of layers during fine-tuning. To further enhance compression, ACNs can be combined with standard pruning techniques—a preliminary investigation of this approach is provided in Appendix G.

5 ACNs *generalize* better

While ACNs demonstrate effective parameter reduction through architectural compression, a key question remains: do these compressed representations offer additional benefits beyond parameter efficiency? In this section, we investigate whether the concentrated information in ACNs' early layers leads to improved generalization capabilities compared to traditional residual architectures. Specifically, we explore robustness to input noise as a proxy for generalization. Moreover, we compare the inherent auto-compression of ACNs—achieved through architectural design—to recent regularization-based methods that rely on externally imposed intermediate losses.

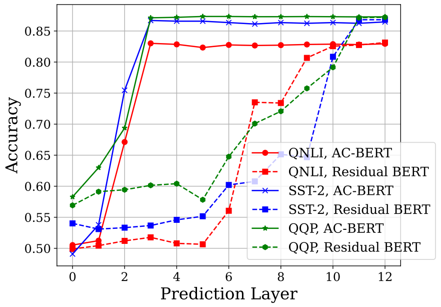


Figure 5: Downstream performance of AC-BERT vs residual BERT on three GLUE tasks.

5.1 Robustness to Input Noise

Next, we present results assessing the robustness of ACNs versus residual transformer architectures to input noise. The experiments are performed with the AC-ViT and residual ViT architectures trained on ImageNet-1K. In this experiment, we inject increasing levels of additive Gaussian noise with standard deviation $\sigma = 0.1, 0.2, 0.4$, and salt-and-pepper noise with percentage of altered pixels $p = 1\%, 2\%, 10\%$. Results (average accuracy) are shown in Table 2 (a) for Gaussian and (b) for salt-and-pepper noise. We observe that *ACNs display improved robustness to noise*, and the performance gap with the residual transformer increases as the noise levels increase. These results align with the findings of [58], who showed that architectures with forward passes closer to feedforward networks (like our ACNs) exhibit enhanced noise robustness. In residual architectures, short connections allow noise to propagate and accumulate throughout the network, whereas the long-connection design of ACNs helps mitigate this amplification effect.

Model	Baseline	Gaussian Noise			Salt and Pepper Noise		
	w/o noise	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.4$	$p = 0.01$	$p = 0.05$	$p = 0.1$
Residual ViT	70.74	67.68	62.80	45.46	56.80	27.48	10.34
AC-ViT	70.76	69.50	64.54	51.89	59.80	36.35	19.98

Table 2: Robustness (average accuracy %) of ViT with long connections (AC-ViT) and with residual connections (Residual ViT) to additive Gaussian noise and salt-and-pepper noise on ImageNet-1K test set.

5.2 Robustness to Data Sparsity

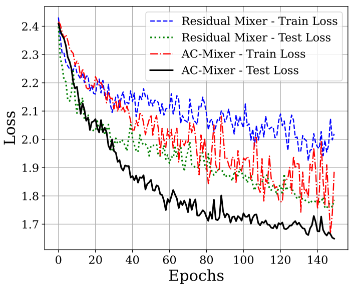


Figure 6: Train and Test Loss of AC-Mixer and Residual Mixer on CIFAR-10 (100 samples per class).

Next, we experimentally compare the performance of residual and long connections architectures in low-data scenarios. For this purpose, we create a random subset of CIFAR-10 [27] by retaining only 100 samples per class, resulting in a total of 1000 examples. Using the same training settings and models as described in Section 4.1 (MLP-Mixer on CIFAR-10), we train both architectures for 150 epochs to assess how fast the training and test loss decrease, as a proxy for the generalization capabilities of each architecture. Results shown in Fig. 6 reveal that ACNs achieve lower training and test loss in fewer epochs compared to residual networks. This faster convergence in loss metrics is a strong indication that auto-compressing networks can be effectively utilized in scenarios with limited data.

6 ACNs *forget* less

Continual learning involves training models on a sequence of tasks without access to past data, aiming to retain performance on previous tasks while learning new ones [8, 55]. A central challenge in CL is catastrophic forgetting—the tendency of neural networks to overwrite old knowledge when updated with new data. Common approaches include data replay methods [39, 33] and regularization techniques that penalize changes to important parameters [26, 61, 2]. We've already demonstrated that ACNs, through implicit layer-wise training, dynamically allocate parameters based on task demands while preserving redundant parameters for future tasks. Conversely, Residual Networks optimized for efficient task learning risk overfitting and suboptimal parameter usage in these sequential learning settings. To test our claims, we evaluate both architectures on the split CIFAR-100 continual learning benchmark, comprising 20 sequential disjoint 5-class classification tasks, focusing on task-incremental learning [51] where task identity is known. We utilize MLP-Mixer architectures (hyperparameters in Appendix B) and we test two continual learning algorithms trained for 10 epochs for each task: naive fine-tuning (Naive FT) and Synaptic Intelligence (SI) [61], which adds a gradient-based regularizer to each parameter depending on how changes in it affect the total loss in a task over the training trajectory. Across experiments, we report Average Forgetting, defined as the mean difference between a task's best performance (right after it is learned) and its final performance after all tasks are learned, and Average Accuracy, defined as the mean accuracy over all tasks at the end of training. We expect gradient-based regularization methods to perform particularly well with ACNs since unused, redundant parameters receive small gradients, making their detection easier compared to Residual Networks where gradients are more uniformly distributed (see gradient heatmaps, Fig. 1(left)). Results in Table 3 confirm our intuition: ACNs consistently exhibit significantly less forgetting (up to 18% improvement) compared to Residual Networks. Notably, with SI, increasing ACN depth decreases forgetting—an ideal behavior for CL systems where increasing network capacity reduces forgetting—while Residual Networks show the opposite pattern, indicating potential overfitting. ACNs also achieve better average accuracy across all tasks, further establishing them as a more suitable architecture for continual learning.

		Avg. Accuracy (%) ↑			**Avg. Forgetting (%) ↓**		
Method	**Arch**	$L = 5$	$L = 10$	$L = 15$	$L = 5$	$L = 10$	$L = 15$
Naive FT	AC-Mixer	32.97 ± 2.4	32.94 ± 5.3	31.61 ± 2.2	46.55 ± 2.2	45.46 ± 5.8	46.91 ± 2.4
	ResMixer	31.77 ± 1.8	28.16 ± 1	26.14 ± 2.3	52.76 ± 2.3	54.89 ± 1.6	54.49 ± 2.2
SI	AC-Mixer	44.5 ± 2.2	46.1 ± 1.3	**46.2 ± 0.8**	35.7 ± 2.1	33.8 ± 0.4	**32 ± 1.8**
	ResMixer	43.47 ± 3.1	36.1 ± 5	32.1 ± 0.8	42.4 ± 4.1	44.6 ± 3.7	50 ± 2.1

Table 3: Average accuracy and forgetting across layers, methods, and architectures on the Split CIFAR-100 continual learning benchmark. Models are trained for 10 epochs per task, where each task consists of classifying 5 out of 100 classes presented sequentially. L denotes the number of layers in the architecture. ACNs consistently **forget less** and they also **do not waste capacity**.

7 ACNs *transfer* better

Parameter redundancy, and specifically potential layer redundancy, in residual architectures is a phenomenon that has been well documented [1, 53, 23]. Recent works [13, 24] have attempted to address this through regularization-based layer-wise compression techniques during training, specifically by adding losses to all intermediate layers of the network and using a weighted sum of them as the total loss, a technique formally introduced in [31] for improved training. Such loss-based regularization methods rely heavily on precise tuning of intermediate loss weights, creating practical challenges. If early-layer loss weights are set too high, the network risks overfitting and poor generalization; if set too low, performance improves gradually across layers with no clear cutoff point, reaching optimal results only at the final layer. This sensitivity to hyperparameter selection makes it difficult to reliably identify an optimal depth for inference using loss-based regularization. ACNs address this challenge through architectural design rather than regularization, naturally compressing information without requiring complex hyperparameter tuning.

To evaluate whether different layer compression approaches learn generalizable representations, we conduct a transfer learning experiment from CIFAR-100 to CIFAR-10 [27] using MLP-Mixer architectures (for hyperparameter choices we refer to Appendix B). This setup allows us to assess how well each model's learned representations transfer to a similar task. In regularization-based layer compression methods, explicitly training all layers to directly minimize a task loss through intermediate supervision can lead to overfitting on the base task, which can further result in weaker transfer capabilities on downstream tasks. In contrast, ACNs' implicit compression mechanism naturally balances generalizability and task performance without imposing external constraints. To ensure fair comparison, we train all models to achieve comparable performance on the CIFAR-100 pre-training task, enabling direct assessment of their transfer capabilities to CIFAR-10. The results in Table 4 confirm our analysis: ACNs demonstrate superior performance on the downstream CIFAR-10 task compared to regularization-based methods, even when upstream CIFAR-100 task performance is similar. This confirms our claim that the representations learned by ACNs are more generalizable and thus exhibit greater transferability.

Method	**Accuracy (%)**
AC-Mixer	**85.38** $\pm$**0.7**
Aligned	82.9 $\pm$0.9
LayerSkip	79 $\pm$1.2

Table 4: C-100 to C-10 transfer learning performance of ACNs vs recent regularization-based layer copression approaches.

8 Conclusion

In this work, we introduced Auto-Compressing Networks (ACNs), an architectural design that organically compresses information into early layers of a neural network during training via long skip connections from each layer to the output, a property we coined as *auto-compression*. Unlike residual networks, ACNs do not require explicit compression objectives or regularization; instead, they leverage architectural design and gradient-based optimization to induce implicit layer-wise training dynamics that drive auto-compression.

Our theoretical and empirical analyses demonstrate that ACNs alter gradient flow, imposing implicit layer-wise training dynamics and resulting in distinct representations compared to feedforward and residual architectures. In practice, this leads to 30–80% of upper layers becoming effectively redundant, enabling faster inference and reduced memory usage without sacrificing accuracy. Experiments across diverse modalities (vision, language) and architectures (ViTs, Mixers, BERT) further show that ACNs match or outperform residual baselines, while offering greater robustness to noise and low-data regimes, excelling in transfer learning, and reducing catastrophic forgetting by up to 18%—all without specialized tuning, overall suggesting that they learn better representations despite using fewer parameters. A summary of the main results is shown in Table 6. Another practical advantage of the auto-compression property of ACNs is its potential to enhance or complement other compression and efficient inference techniques. Preliminary results demonstrating this synergy are presented in Appendices G and H.

Concluding, ACNs highlight the potential of architectural design as implicit regularization and pave the way toward self-adapting neural networks that allocate depth and capacity dynamically. Future work may extend ACNs to self-supervised, multi-task, and generative settings, as well as explore per-sample adaptive inference and other short vs long architectural variants. Some initial

9 Limitations and Broader Impact

Due to resource constraints, our evaluation of ACNs was limited to relatively small-scale tasks, though the architecture consistently performed well across modalities, datasets, and state-of-the-art baselines. Broader validation—including large-scale, self-supervised, and multi-task settings (e.g., language or multimodal models)—is essential to fully understand its capabilities and boundaries. A notable limitation is the increased training time compared to residual architectures. While this may contribute to stronger representations, optimizing training efficiency remains an open challenge, warranting further exploration of scheduling and initialization strategies.

By enabling implicit layer pruning, ACNs aim to reduce inference-time resource use, supporting more sustainable AI. Long-term, this could inspire adaptive architectures with System 1 / System 2 behavior [25], dynamically adjusting depth based on task complexity. However, like all efficiency advances, broader deployment may introduce ethical concerns—such as misuse in low-regulation environments or privacy-sensitive applications—highlighting the need for responsible development and oversight.

10 Acknowledgments and funding disclosure

This work has been partially supported by project MIS-5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the EU under the NextGenerationEU Program. We acknowledge EuroHPC JU project ID EHPC-AI-2024-A04-051 for use of the supercomputer LEONARDO@ CINECA, Italy.

G. Paraskevopoulos was funded by the European Union's Horizon Europe research and innovation programme under the AIXPERT project (Grant Agreement No. 101214389).

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018.

[3] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, 2021.

[4] Yu Bai, J. Zico Kolter, and Vladlen Koltun. Understanding the low-rank bias of deep neural networks. In *International Conference on Learning Representations*, 2021.

[5] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *International conference on machine learning*, pages 342–350. PMLR, 2017.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[7] Róbert Csordás, Christopher D Manning, and Christopher Potts. Do language models use their depth efficiently? *arXiv preprint arXiv:2505.13898*, 2025.

[8] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[10] 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. In *International Conference on Learning Representations*, 2021.

[11] Zi-Yi Dou, Zhaopeng Tu, Xing Wang, Shuming Shi, and Tong Zhang. Exploiting deep representations for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4253–4262, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[12] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 12622–12642. Association for Computational Linguistics, 2024.

[13] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[14] Muhammad ElNokrashy, Badr AlKhamissi, and Mona Diab. Depth-wise attention (dwatt): A layer fusion method for data-efficient classification. *arXiv preprint arXiv:2209.15168*, 2022.

[15] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in neural information processing systems*, 2, 1989.

[16] Kirsten Fischer, David Dahmen, and Moritz Helias. Field theory for optimal signal propagation in resnets. *arXiv preprint arXiv:2305.07715*, 2023.

[17] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[18] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[19] Suriya Gunasekar, Jason Lee, Tong Zhang, and Behnam Neyshabur. Implicit bias of gradient descent on wide deep networks. In *Advances in Neural Information Processing Systems*, 2018.

[20] 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.

[21] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2017.

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[23] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 646–661. Springer, 2016.

[24] Jiachen Jiang, Jinxin Zhou, and Zhihui Zhu. Tracing representation progression: Analyzing and enhancing layer-wise similarity. In *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Daniel Kahneman. Thinking, fast and slow. *Farrar, Straus and Giroux*, 2011.

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[29] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations*, 2017.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[31] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570. Pmlr, 2015.

[32] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[33] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[34] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[35] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[36] Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. In *International Conference on Learning Representations*, 2018.

[37] Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[39] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[40] Maria Refinetti, Stéphane d'Ascoli, Ruben Ohana, and Sebastian Goldt. Align, then memorise: the dynamics of learning with feedback alignment. In *International Conference on Machine Learning*, pages 8925–8935. PMLR, 2021.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[42] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

[43] Pedro HP Savarese, Leonardo O Mazza, and Daniel R Figueiredo. Learning identity mappings with residual gates. *arXiv preprint arXiv:1611.01260*, 2016.

[44] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling, 2022.

[45] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[47] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[48] Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25219–25227, 2025.

[49] 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.

[50] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

[51] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[52] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[53] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[54] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[55] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[56] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[57] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] Zonghan Yang, Yang Liu, Chenglong Bao, and Zuoqiang Shi. Interpolation between residual and non-residual networks. In *International Conference on Machine Learning*, pages 10736–10745. PMLR, 2020.

[59] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

[60] Alireza Zaeemzadeh, Nazanin Rahnavard, and Mubarak Shah. Norm-preservation: Why residual networks can become extremely deep? *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3980–3990, 2020.

[61] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR, 2017.

[62] Xiao Zhang, Ruoxi Jiang, William Gao, Rebecca Willett, and Michael Maire. Residual connections harm generative representation learning. *arXiv preprint arXiv:2404.10947*, 2024.

[63] Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations*, 2025.

[64] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main scope/contribution of the paper that is stated in the abstract and introduction is the introduction of an alternative to residual networks architecture, theoretically and empirically analyzed throughout the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide a "Limitations" Section in the main paper.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.

- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: In our theoretical results we provide assumptions (aka 1d linear analysis of gradient equations) and analytical derivations can be found in Appendix.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We clearly describe experiments and evaluation setup and we further provide all experimental details in the Appendix. Moreover, we provide an anonymous link where code for the paper will be released.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may

be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide an anonymous link where code for the paper will be released. Datasets used in the paper are publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: In the main paper we discuss core details regarding experiments and in the Appendix we report detailed choices.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer:[Yes]

   Justification: We report average accuracy where applicable (e.g., noise experiments), and include mean metrics with standard deviation in continual learning settings where variability is significant.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Our experiments are small-scale but we consistently report training epochs.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer:[Yes]

We conform to the NeurIPS Code of Ethics. We clearly describe our methods, include details to reproduce them and plan to release our code.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We include a "Broader Impact" section.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer:[Yes]

Justification: We do not release new data and we train small-scale discriminative models.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We check the license and cite models and datasets we use in the experiments.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We train small reproducible models with public data and explain in detail our setups. We do not introduce major assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: not invloved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: not involved

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLMs are only used for grammar and formatting.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Technical Appendices and Supplementary Material

A Extended Literature Review

We present here an extended literature review across four key areas relevant to our work: 1) residual connections and their role in training stability, 2) architectural variants with longer/denser residual connections, 3) methods that employ intermediate layer losses to learn better representations and exit early, and 4) neural architectures that induce regularization and better representations. Our work is more closely related to research area 4, but it is important to note that training stability, efficiency, performance, and representation learning are related goals, which can be achieved either through architectural choices (1, 2, 4) and / or through additional optimization criteria (3).

Residual Connections and Training Stability: Training deep neural networks with gradient descent becomes increasingly difficult as network depth increases. Multipath network architectures date back to the 1980s, with early work exploring cascade structures in fully connected networks trained layer by layer to improve training stability [15]. Highway networks [47] introduced gated bypass paths that allowed for effective training of networks with hundreds of layers. [20] found that deeper convolutional neural networks (CNNs) not only suffer from a decrease in generalization performance, often due to overfitting, but also experience a decrease in training performance. To address this, they introduced residual connections (or identity mappings), proposing that learning residual functions relative to identity mappings simplifies optimization. These skip connections improve the training process and often improve performance ([5], [36], [60], [32]). [53] further argued that a residual network with n layers can be viewed as a collection of 2^n paths of varying length. At each layer, the signal either skips the layer or passes through it, creating 2^n possible paths. Despite sharing weights, these paths function as an ensemble of networks, as confirmed by experiments. In contrast, a traditional deep feedforward network has only one path, so removing any random layer significantly degrades performance. Additionally, the authors showed that these paths are typically shallow, with backward gradients often vanishing after passing through only a small fraction of the total layers.

Residual Variants: Following the success of residual networks various architectural modifications were proposed to improve efficiency and performance. In DenseNets [22], each layer is connected to all subsequent layers enabling for more efficient feature reuse; fusion is achieved through concatenation rather than addition. More recently, DenseFormer [37] introduced learned weighted averaging across layer outputs, while Depth-Wise Attention [14] applies attention mechanisms across block outputs.

Intermediate Supervision and Early Exit: In deeply supervised nets [31], complementary objectives are added to all intermediate layers to encourage hidden layers to learn more discriminative representations. In this approach, each intermediate objective i is a loss function that captures the classification error of an SVM trained on the output features of layer i. The overall loss is the sum of the intermediate and final objectives. This idea evolved in several directions: Graves [17] proposed adaptive computation time, while more recent work like MSDNet [21] and CALM [44] introduced dedicated prediction heads. Other approaches employ trainable routing mechanisms [56, 57] to determine layer usage. Concurrent to our work, LayerSkip [12] proposes an architecture similar to ACNs, focusing primarily on inference acceleration through layer dropout and early exit mechanisms. Additionally, [24] also incorporates intermediate losses with a common head and a linearly increasing weight curriculum, justifying it through the lens of representational similarity between intermediate layers. While these approaches rely on explicit auxiliary objectives or dedicated components, our work achieves similar benefits through architectural design alone, enabling natural depth determination through gradient-based optimization.

Architecturally-induced regularization and representation learning: Stochastic regularization methods like Dropout [46] and its variants demonstrated that randomly dropping connections during training can lead to more robust feature learning. This insight was extended to other structural approaches like Stochastic Depth [23] where randomly dropping entire layers improved generalization. Residual connections initially proposed to address the vanishing gradient problem [20] have been shown to contribute to smoother loss landscapes and improved generalization [32]. These findings align with theoretical work showing that architectural choices impose implicit biases that influence the solutions found during training [19]. Recent work on transformers shows that architectural choices like attention patterns and layer normalization can also induce implicit regularization effects, e.g., the combination of skip connections and layer normalization can bias the model toward low-rank solutions

[4]. The proposed long connection approach builds on these insights, using architectural design to naturally encourage the learning of robust representation while enabling automatic information compression allowing for early exit.

B Experimental Details and Setup

CIFAR-10 - MLP Mixer: The MLP Mixers have 16 layers with a hidden size of 128. The patch size is 4 (the input is 32x32, 3 channels). The MLP dimension D_C is 512, while D_S is 64. We are using the AdamW optimizer [34] with a maximum learning rate of 0.001 and a Cosine Scheduler with Warmup. The batch size is 64.

BERT post-training pruning: For Magnitude pruning, we consider the setting where the pruning happens after fine-tuning on the downstream task. For Movement pruning, we follow a gradual fine-tune and prune curriculum, where in setting (I): 20% of the parameters are pruned after each epoch, whereas in setting (II): we prune 40% of the parameters after an epoch.

Continual Learning Experiments: We are using the same MLP-Mixer setup with the CIFAR-10 experiment (see above). We train for 10 epochs in each task, using AdamW with learning rate of 0.001 and a batch size of 64. For Synaptic Intelligence we use a coefficient $\lambda = 1$.

Transfer Learning experiment Hyperparameter Choices: In our main paper experiment we compare: 1) our proposed AC-Mixer, 2) a Residual Mixer with the setup of [24] (Aligned) and 3) a Residual Mixer with the setup of [13] (LayerSkip), with the rotational early exit curriculum with $p_{max} = 0.1$, $e_{scale} = 0.2$.

C Gradient Propagation equations derivation

Notation: x_i is the output of layer i, w_i is the weight of layer i (the weight used to construct x_i), x_0 is the input (after a potential initial embedding operation) and y_F, y_R, y_A is the output for each architecture.

FFN forward pass

$$y_F(= x_L) = \prod_{i=1}^{L} w_i x_0 \tag{2}$$

FFN backward pass for weight i

$$\frac{\partial y_F}{\partial w_i} = \frac{\partial y_F}{\partial x_i}\frac{\partial x_i}{\partial w_i} \tag{3}$$

$$\frac{\partial y_F}{\partial w_i} = \underbrace{\left(\prod_{k=i+1}^{L} w_k\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1} w_m\right)}_{\textit{forward}\text{ term}} x_0 \tag{4}$$

ResNet forward pass

$$x_i = w_i x_{i-1} + x_{i-1} = (1 + w_i)x_{i-1} \tag{5}$$

$$y_R = \prod_{i=1}^{L}(1 + w_i)x_0 \tag{6}$$

ResNet backward pass for weight i

$$\frac{\partial y_R}{\partial w_i} = \frac{\partial y_R}{\partial x_i}\frac{\partial x_i}{\partial w_i} = \left(\prod_{k=i+1}^{L}(1 + w_k)\right)\left(\prod_{m=1}^{i-1}(1 + w_m)\right)x_0 \tag{7}$$

$$\frac{\partial y_R}{\partial w_i} = \underbrace{\left(1 + \sum_{k=i+1}^{L} w_k + \sum_{i+1 \le k < j \le L} w_k w_j + \cdots + \prod_{k=i+1}^{L} w_k\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1}(1+w_m)\right)}_{\textit{forward}\text{ term}} x_0 \tag{8}$$

or equivalently:

$$\frac{\partial y_R}{\partial w_i} = \underbrace{\left(1 + \sum_{k=1}^{L-i+1} \texttt{sum of } \binom{L-i+1}{k} w \texttt{ k-tuples}\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1}(1+w_m)\right)}_{\textit{forward}\text{ term}} x_0 \tag{9}$$

ACN forward pass

$$y_A = x_0 + \sum_{i=1}^{L} x_i = x_0 + \sum_{i=1}^{L} \prod_{j=1}^{i} w_j x_0 \tag{10}$$

ACN backward pass for weight i

$$\frac{\partial y_A}{\partial w_i} = \frac{\partial y_A}{\partial x_i}\frac{\partial x_i}{\partial w_i} = \left(1 + \sum_{k=i+1}^{L} \frac{\partial x_k}{\partial x_i}\right) x_{i-1} \tag{11}$$

$$\frac{\partial y_A}{\partial w_i} = \underbrace{\left(1 + \sum_{j=i+1}^{L} \prod_{k=i+1}^{j} w_k\right)}_{\textit{backward}\text{ term}} \underbrace{\left(\prod_{m=1}^{i-1} w_m\right)}_{\textit{forward}\text{ term}} x_0 \tag{12}$$

D Analysis of the Connectivity across architectures

Here, we further clarify the shown connections on Table 1. The red connections shown for ResNet connectivity, show the implicit direct connections (paths) that are formed in the architecture through the residual summation. Specifically, we denote:

- $z_0 = x_0$
- z_{i-1}: input to layer i (for $i > 0$), which is multiplied with w_i to get:
- $x_i = w_i \cdot z_{i-1}$: output of layer i **before** residual summation
- $z_i = z_{i-1} + x_i$: output of layer i **after** residual summation, or equivalently, input to layer $i+1$

In our case, we want to express the input to the final layer z_2, which is:

$$\begin{aligned} z_0 &= x_0 \\ z_1 &= x_1 + z_0 = x_1 + x_0 \\ z_2 &= x_2 + z_1 = x_2 + x_1 + x_0 \end{aligned}$$

Thus, we see that the residual summation $z_i = z_{i-1} + x_i$ effectively results in each layer i receiving as input (z_i) a cumulative sum of all previous layer outputs (x_*). In standard feedforward networks (FFNs), each layer directly receives input only from the immediate previous layer. In contrast, in ACNs, intermediate layers behave like those in FFNs, but the final layer receives as input the outputs of all preceding layers.

E Comparisons with Other Residual Architectures

To address concerns of overfitting and ensure fair comparisons in terms of training time, we hypothesize that ACNs tend to train longer due to a reduced number of information pathways compared to residual architectures (which scale linearly vs. exponentially in terms of connectivity). However, rather than overfitting, we argue that ACNs utilize this extended training period to learn more robust and generalizable representations.

To verify this, we conducted a controlled experiment using various architectures in the MLP-Mixer setting on CIFAR-10, training all models for 700 epochs to ensure fair comparison. The evaluated models include:

- **AC-Mixer:** An auto-compressing (ACN-style) Mixer,
- **Res-Mixer:** A vanilla residual Mixer,
- **DenseNet-Mixer:** A DenseNet-style variant [22], where each layer receives as input the concatenation of all previous layers' outputs, projected back to the hidden dimension via a learnable projection matrix. In order to have similar parameter count with the other architectures (since this architecture introduces additional parameters growing with depth), we reduce the number of layers to 10 (compared to 16 in the other models), and
- **DenseFormer-Mixer:** A DenseFormer-style variant [37], where each layer receives a learnable linear combination of all previous layers' outputs, with one scalar weight per previous layer ($O(L^2)$ extra parameters).

We report: (1) **Accuracy**, (2) **Best Acc. Epoch** — the epoch at which this peak accuracy was reached; and (3) **Cutoff Layer** — the earliest layer whose output reaches comparable performance to the final output, to measure auto-compression.

Model	Accuracy	%Network used	Best Acc. Epoch
Res-Mixer	90.3 ± 0.09	100%	615
DenseNet-Mixer	90.3 ± 0.10	100%	600
DenseFormer-Mixer	90.4 ± 0.09	100%	637
AC-Mixer	**92.0** ± 0.08	75%	695

Table 5: Performance comparison across AC and residual variants on the MLP-Mixer/CIFAR-10 experiment. AC-Mixer converges slower but achieves the highest accuracy (improved generalization) with an earlier cutoff layer, showcasing stronger auto-compression.

The results in Table 5 show that AC-Mixer converges more slowly than residual variants but leverages the extended training to learn more compact and generalizable representations. Unlike residual and densely connected variants, which reach peak performance earlier, AC-Mixer achieves higher accuracy with fewer effective layers (earlier cutoff) and without signs of overfitting but rather stronger generalization, supporting our hypothesis.

F Summary of Results

Table 6 presents the main results comparing residual (Res-) and auto-compressing (AC-) variants across both vision and language tasks. We observe that ACN variants consistently achieve comparable or superior accuracy while requiring significantly fewer parameters, lower computational cost at inference, and reduced storage requirements. These findings highlight the efficiency and scalability of the AC design, making it a strong alternative to traditional residual architectures.

G Post-Training Pruning with AC-Encoders

ACN's primary advantage lies in its inherent compression capabilities during training, suggesting that when combined with pruning techniques, it should significantly outperform traditional residual architectures. To provide validation for this hypothesis, we conducted experiments using magnitude

Model	**Accuracy ↑**	**#Params ↓**	**GFLOPs ↓**	**Size (MB) ↓**
Res-ViT on ImageNet	70.74 ± 0.09	86M	33.7	330
AC-ViT on ImageNet	70.76 ± 0.12	51M	19.7	195
Res-BERT on SST-2	86.63 ± 0.09	110M	21.72	418
AC-BERT on SST-2	86.68 ± 0.06	46M	5.44	174
Res-BERT on QNLI	83.14 ± 0.07	110M	21.72	418
AC-BERT on QNLI	83.07 ± 0.10	46M	5.44	174
Res-BERT on QQP	87.20 ± 0.09	110M	21.72	418
AC-BERT on QQP	87.30 ± 0.07	46M	5.44	174

Table 6: Comparison of residual (Res-) and auto-compressing (AC-) variants across both vision and language tasks.

and movement pruning [42], two commonly employed baseline pruning techniques. Results are shown when fine-tuning of the SST-2 dataset sentiment analysis task. We refer to Appendix B for details regarding the experimental setup. Figure 7(right) confirms our hypothesis: ACNs consistently demonstrate superior compression-performance trade-offs compared to standard architectures, with their advantage becoming more pronounced at higher compression rates. This indicates that ACNs' architectural design naturally leads to more efficient parameter utilization, creating representations that are inherently more amenable to further pruning. While these preliminary results validate our approach to addressing parameter redundancy, they also point toward promising future directions. We anticipate that combining pre-trained ACN architectures with state-of-the-art pruning methods will result in extremely efficient, high-performing models, though rigorous validation of this hypothesis requires further investigation.

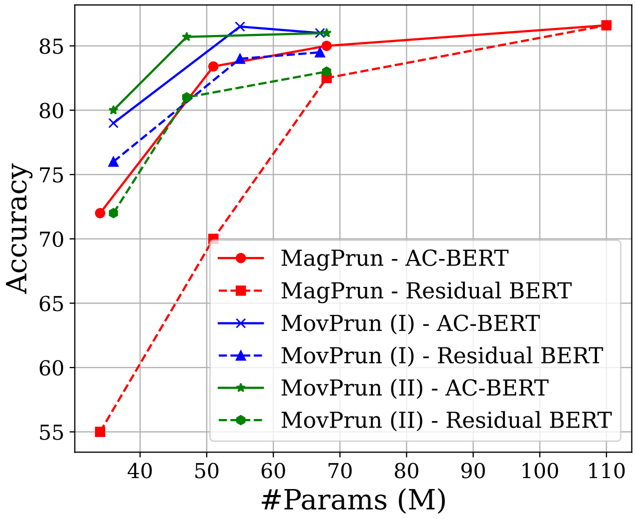


Figure 7: Accuracy vs model size of AC-BERT and Residual BERT on SST-2 when pruned with Magnitude and Movement Pruning (with two different settings, refer to Appendix B for details).

H Relation to Early Exit (EE) Methods

In this section, we explore the relationship between ACNs and Early Exit (EE) methods. While both aim to improve computational efficiency, ACNs are fundamentally different in their approach. Traditional EE methods are training-time strategies that rely on *explicit intermediate supervision*, whereas ACNs are architectural designs that achieve similar benefits *implicitly*, without auxiliary losses or supervision. Importantly, these approaches are complementary: ACNs can be combined with EE techniques to further enhance both efficiency and flexibility.

Overview of EE Methods. Early Exit methods (e.g., BranchyNet [49]) typically:

1. Introduce explicit losses at intermediate layers to encourage early discriminative features.
2. Train using a weighted combination of intermediate and final losses.
3. Use confidence-based criteria (e.g., entropy) at inference to decide whether to exit early.

This line of work, originating from Deep Supervision [31], introduces several hyperparameters—such as loss weights—that require careful tuning. This tuning is crucial for the final behavior of the network since overemphasizing early-layer losses can lead to overfitting, as early layers are pushed to achieve low training loss.

In contrast, ACNs do not use any intermediate losses or auxiliary heads. They are trained end-to-end like standard feedforward or residual networks. Nevertheless, ACNs naturally produce more informative intermediate representations due to their implicit layer-wise compression dynamics. As demonstrated in Section 7, adding intermediate supervision to residual networks requires delicate balancing to avoid overfitting, while ACNs generalize more robustly—particularly in transfer settings—without any such tuning, thanks to their architectural regularization.

ACNs complement EE methods. Given that ACNs naturally concentrate discriminative information in earlier layers, they are well-suited for integration with EE mechanisms. To validate this, we conducted experiments on CIFAR-10 using MLP-Mixer variants, comparing six configurations:

- **Res-Mixer:** Standard residual MLP-Mixer,
- **AC-Mixer:** auto-compressing MLP-Mixer,
- **Res-Mixer w/ Branches:** Explicit early-exit branches at layers 4, 8, and 12, each with a dedicated MLP classifier,
- **AC-Mixer w/ Branches:** Same branching structure applied to the AC-Mixer,
- **Res-Mixer w/ Shared EE Head:** All layers connect to a shared classifier head, following recent works [24, 13], allowing exit decisions without additional parameters, and
- **AC-Mixer w/ Shared EE Head:** Same shared-head mechanism applied to the AC-Mixer.

For the **Branches** method, we use fixed weights $[0.8, 0.6, 0.4]$ for the three intermediate branches. In contrast, the **Shared EE Head** employs a linearly increasing layer weighting scheme, as in [24], defined by:

$$w_l = \frac{2(l+1)}{L(L+1)}, \quad \text{for } l = 0, 1, \ldots, L-1.$$

During inference, we compute the entropy of the logits at each potential exit point. If the entropy falls below a pre-defined threshold ($\tau = 0.7$ for the results below), the model exits at that layer. Inference speedup is measured in terms of FLOPs, normalized to the baseline Res-Mixer.

Model	**Speedup (FLOPs)**
Res-Mixer	1.0×
AC-Mixer (global cutoff at $L = 12$)	1.5×
Res-Mixer w/ Branches	2.2×
AC-Mixer w/ Branches	2.6×
Res-Mixer w/ Shared EE Head	2.4×
AC-Mixer w/ Shared EE Head	**3.3**×

Table 7: Inference speedup (measured in FLOPs) of various Early Exit configurations on CIFAR-10 using MLP-Mixer variants. All values are normalized to the FLOPs of the full Res-Mixer.

The results (Table 7) confirm that ACNs substantially improve the effectiveness of EE methods. When combined with a shared early-exit head, the AC-Mixer achieves the highest speedup—more than

3×—benefiting from both implicit compression and confident early exits. This synergy highlights the strength of ACNs as a foundation for efficient inference.

We also draw a parallel between this behavior and the compatibility of ACNs with pruning methods (see Appendix G): in both cases, the compact representations of ACNs, through auto-compression, facilitate downstream efficiency techniques. Developing new early-exit strategies tailored to ACNs is a promising direction for future work.