# Interpolated-MLPs: Controllable Inductive Bias

**Sean Wu**[*] SEANWU@ETHZ.CH
**Jordan Hong**[*] JOHONG@ETHZ.CH
**Keyu Bai**[*] KEYBAI@ETHZ.CH
**Gregor Bachmann** GREGORB@ETHZ.CH
*ETH Zurich, Switzerland*

## Abstract

Due to their weak inductive bias, Multi-Layer Perceptrons (MLPs) have subpar performance at low-compute levels compared to standard architectures such as convolution-based networks (CNN). Recent work, however, has shown that the performance gap drastically reduces as the amount of compute is increased without changing the amount of inductive bias [1]. In this work, we study the converse: in the low-compute regime, how does the incremental increase of inductive bias affect performance? To quantify inductive bias, we propose a "soft MLP" approach, which we coin *Interpolated MLP* (I-MLP). We control the amount of inductive bias in the standard MLP by introducing a novel algorithm based on interpolation between fixed weights from a *prior model* with high inductive bias. We showcase our method using various *prior models*, including CNNs and the MLP-Mixer architecture. This interpolation scheme allows fractional control of inductive bias, which may be attractive when full inductive bias is not desired (e.g. in the mid-compute regime). We find experimentally that for Vision Tasks in the low-compute regime, there is a continuous and two-sided logarithmic relationship between inductive bias and performance when using CNN and MLP-Mixer prior models.

## 1. Introduction

MLPs provide a rich theoretical deep learning framework due to their mathematical simplicity despite inferior practical performance. For vision tasks, modern models based on the Convolutional Neural Network (CNN) [7] [5] and Vision Transformer (ViT) [4] have demonstrated better performance. A recent hypothesis claims that CNN and ViT are superior to MLPs because inductive bias is detrimental at high-compute scales [4].

The work of Bachmann et al. [1] recently strengthened this hypothesis and showed that with sufficient parameter count and pre-training, even simple MLPs can achieve strong performance on vision tasks. We aim to continue this line of research and ask: i) at a low-compute scale, what is the relationship (e.g. linear, logarithmic, exponential) between the amount of inductive bias and performance? ii) since there is a balance of tension (inductive bias is useful in low-compute but not desired in high-compute), can we parameterize (and optimize over) the fractional amount of inductive bias, particularly in the mid-compute regime?

In this study, we introduce inductive bias to MLPs by methods of *locality injection*, first proposed by Ding et al. [3], where they converted a convolution kernel $F$ into a fully-connected layer $\mathbf{W_F}$, then merged (adding) $\mathbf{W_F}$ to a parallel fully-connected layer $\mathbf{W}$. The inductive bias is drawn from the convolutional kernel, which we will refer to as the *prior model*.

In this study, we extend the locality injection approach in two directions. First, instead of adding the prior model to the MLP layer, we interpolate using a parameter $\alpha$ to fractionally control the amount

[*]. Equal contribution.

of added inductive bias. Second, in addition to a CNN, we extend the interpolation to another prior model: the MLP-Mixer [8]. For both prior models, we i) investigate the trade-off between the amount of inductive bias $\alpha$ and performance, and ii) provide architectures and training methods that are performant under compute constraints.

## 2. Background

**Notation.** We consider an arbitrary layer with tensor input $\mathbf{x} \in \mathbb{R}^{c_{in} \times h_{in} \times w_{in}}$ and output $\mathbf{y} \in \mathbb{R}^{c_{out} \times h_{out} \times w_{out}}$ , where $c_{in}, h_{in}, w_{in}$ denote the channel, height, and width of the input, $c_{out}, h_{out}, w_{out}$ denote the same for the output. For MLP layers, the tensor input and output are vectorized into a single dimension: $vec(\mathbf{x}) \in \mathbb{R}^{c_{in}h_{in}w_{in}}$ and $vec(\mathbf{y}) \in \mathbb{R}^{c_{out}h_{out}w_{out}}$. The equivalent fully-connected layer representation of a *prior model* is denoted as $\mathbf{W_P} \in \mathbb{R}^{c_{out}h_{out}w_{out} \times c_{in}h_{in}w_{in}}$.

**Standard MLP.** As a starting point, we consider the standard MLP (S-MLP) as in [1] with an input $\mathbf{z}$ of general dimension $M_{in}$, weight matrix $\mathbf{W} \in \mathbb{R}^{M_{out} \times M_{in}}$, nonlinear activation $\sigma$, and Layer Normalization LN :

$$\text{Block}(\mathbf{z}) = \sigma\left(\text{LN}(\mathbf{z}\mathbf{W}^T)\right). \tag{1}$$

For vision tasks, the tensor input and output are vectorized, i.e. $\mathbf{z} = vec(\mathbf{x})$, $M_{in} = c_{in}h_{in}w_{in}$, and $M_{out} = c_{out}h_{out}w_{out}$.

$$vec(\mathbf{y}) = vec(\mathbf{x})\mathbf{W}^T. \tag{2}$$

The flattening procedure removes all locality of an image tensor. As such, the MLP does not possess any inductive bias for vision tasks. However, such locality is particularly relevant to images. Next, we introduce the two prior models: CNN and MLP-Mixer.

**CNN.** Convolution can be viewed as a special case of an MLP layer, in which a special weight matrix $\mathbf{W_F}$ is structured by being sparse and having shared weights, which leads to spatially localized learning and introduces inductive bias useful for vision tasks [1]. We use $F$ to denote the convolution kernel and express the output tensor as:

$$\mathbf{x}^{(out)} = F * \mathbf{x}^{(in)}. \tag{3}$$

Equation (3) can be expressed using the MLP formulation with vectorized input and output.

$$vec(\mathbf{x}^{(out)}) = vec(F * \mathbf{x}^{(in)}) = vec(\mathbf{x}^{(in)})\mathbf{W_F}^T, \tag{4}$$

where $\mathbf{W_F} \in \mathbb{R}^{c_{out}h_{out}w_{out} \times c_{in}h_{in}w_{in}}$ is the equivalent fully-connected layer obtained from $F$. Here, $\mathbf{W_P} = \mathbf{W_F}$. Appendix A shows the explicit construction from $F$ to $\mathbf{W_F}$.

**MLP-Mixer.** We identify three sources of inductive bias in the MLP-Mixer, all of which we model as a special case of a fully-connected layer: i) Patchifying and per-patch linear embeddings, ii) Token-mixing with weight sharing, iii) Channel-mixing with weight sharing. These inductive biases allow the MLP-Mixer to attain competitive results and perform comparably to the current state-of-the-art ViT [8]. Collectively, the above operations can be expressed as two successive operations: a linear operation $\mathbf{L} \in \mathbb{R}^{c_{in}h_{in}w_{in} \times c_{in}h_{in}w_{in}}$, and a Toeplitz matrix $\tilde{\mathbf{W}} \in \mathbb{R}^{c_{out}h_{out}w_{out} \times c_{in}h_{in}w_{in}}$ acting on the vectorized input $vec(\mathbf{x})$. The linear operation $\mathbf{L}$ can be either a patchifying matrix $\mathbf{P}$, a transpose matrix $\mathbf{T}$, or an identity matrix $\mathbf{I}$ in the degenerate case.

$$vec(\mathbf{y}) = vec(\mathbf{x})\mathbf{L}^T\tilde{\mathbf{W}}^T \tag{5}$$

The fully-connected equivalent matrix is given by $\mathbf{W_P} = \tilde{\mathbf{W}}\mathbf{L}$ and derived in Appendix B .

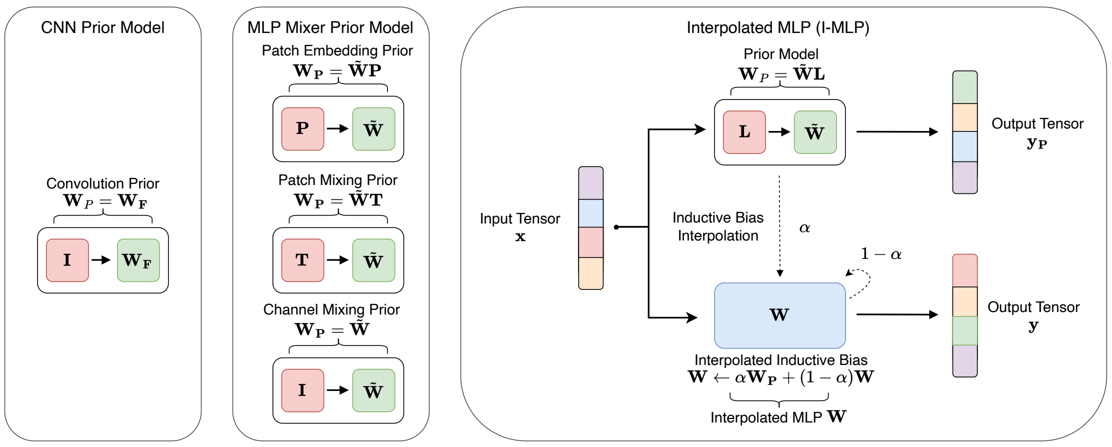

CNN Prior Model
Convolution Prior
$\mathbf{W}_P = \mathbf{W_F}$
$\mathbf{I}$
$\mathbf{W_F}$
MLP Mixer Prior Model
Patch Embedding Prior
$\mathbf{W_P} = \tilde{\mathbf{W}}\mathbf{P}$
$\mathbf{P}$
$\tilde{\mathbf{W}}$
Patch Mixing Prior
$\mathbf{W_P} = \tilde{\mathbf{W}}\mathbf{T}$
$\mathbf{T}$
Channel Mixing Prior
$\mathbf{W_P} = \tilde{\mathbf{W}}$
Interpolated MLP (I-MLP)
Prior Model
$\mathbf{W}_P = \tilde{\mathbf{W}}\mathbf{L}$
$\mathbf{L}$
Output Tensor
$\mathbf{y_P}$
Input Tensor
$\mathbf{x}$
Inductive Bias Interpolation
$\alpha$
$1-\alpha$
$\mathbf{W}$
Output Tensor
$\mathbf{y}$
Interpolated Inductive Bias
$\mathbf{W} \leftarrow \alpha\mathbf{W_P} + (1-\alpha)\mathbf{W}$
Interpolated MLP $\mathbf{W}$

Figure 1: Graphic description of the Interpolated MLP (left), the CNN prior (middle) and the MLP-Mixer prior (right).

## 3. Interpolation Training Method

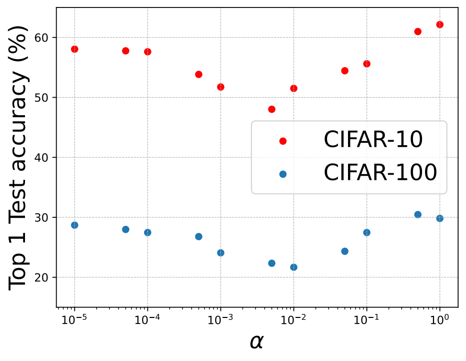
Algorithm 1 outlines the training method for our proposed Interpolated MLP (I-MLP), which interpolates with a prior model layer in every epoch. For a multi-layer network, the update rule in Algorithm 1 is applied independently to all layers. In this paper, we study the interpolation for two prior models: CNN and MLP-Mixer. We obtain $\mathbf{W_P}$ using Equation 4 for CNNs and Equation 5 for MLP-Mixers.

```
Algorithm 1: Interpolated MLP (I-MLP) training method
Initialize: I-MLP weight matrix $\mathbf{W}$ (fully-connected) and prior model $P$
Training:
foreach epoch do
    Independently train $\mathbf{W}$ and $P$ (forward and back propagation).
    Interpolate $\mathbf{W}$: $\mathbf{W} \leftarrow (1-\alpha)\mathbf{W} + \alpha\mathbf{W_P}$, where $\mathbf{W_P}$ is the converted fully-connected
      layer from the prior model $P$.
end
```

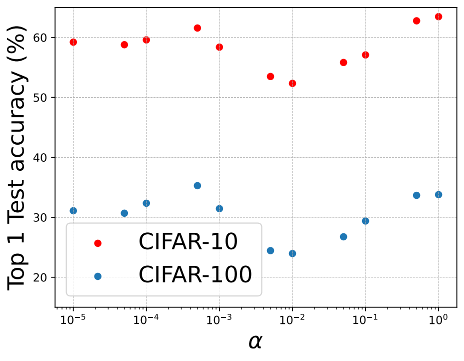
When $\alpha = 1$, the interpolated $\mathbf{W}$ is equivalent to the prior model $P$. When $\alpha = 0$, the interpolated $\mathbf{W}$ receives no inductive bias and is just a pure fully-connected layer with no structural constraints. By adjusting $\alpha$, we can control the amount of inductive bias introduced into the MLP.

## 4. Experiments

We investigate the impact of interpolated inductive bias with several experiments. For the two prior models (CNN and MLP-Mixer), we design parallel I-MLP structures. We then train the pair of models (prior model, I-MLP) according to Section 3 and vary the value of $\alpha$ between 0 and 1. Our Interpolated-MLP model follows the Standard MLP (S-MLP) architecture described by Bachmann et al. [1]. The experimental setup can be found in Appendix E.



(*a*) Prior model: CNN



(*b*) Prior model: MLP-Mixer

Figure 2: Test accuracy of I-MLP with varying $\alpha$ (with data augmentation).

### 4.1. Interpolation with Varying $\alpha$

To interpolate between a CNN and an MLP, we use the Standard CNN (S-CNN) based on AlexNet [7] as our prior model. The CNN has six convolutional layers, each with an output dimension of $d_{CNN} = o\,h_{out}\,w_{out}$. The I-MLP (with CNN prior) has six fully-connected layers, each with a width of $d_{MLP} = 1024$. To ensure proper interpolation, we meticulously match the layer dimensions between the pair; at each layer, $d_{CNN} = d_{MLP}$. The detailed method and exact layer dimensions are outlined in Appendix C.

Similarly, for interpolating between an MLP-Mixer and an MLP, we use the original MLP-Mixer from [8] as our prior model, with specific dimensions outlined in the appendix. We design the I-MLP (with MLP-Mixer prior) by replacing the three sources of inductive bias with fully-connected layers. The interpolation method is detailed in Sections B.1 and B.2, and the architectures of the two models are described in Appendix E (Table 3).

We vary the interpolation weight $\alpha$ and plot the corresponding top 1 test accuracies on a semilogarithmic plot in Figure 2 with a CNN prior (left) and MLP-Mixer prior (right). In Figure 2 (left), we observe a minimum at around $\alpha = 5 \times 10^{-3}$, with either side of the minimum exhibiting logarithmic behavior. We hypothesize that the S-MLP and S-CNN models are converging to different local minima in the loss landscape; when we interpolate between them, there exists an $\alpha$ such that the two minima interfere with equal strength, resulting in poor performance. Decreasing $\alpha$ below $5 \times 10^{-3}$ improves performance logarithmically, approaching S-MLP levels. Similarly, increasing $\alpha$ above $5 \times 10^{-3}$ improves performance logarithmically, approaching S-CNN levels. In Figure 2 (right), the same logarithmic behavior is observed for the I-MLP with MLP-Mixer prior. However, we observe both a minimum at $\alpha = 10^{-2}$ and a (local) maximum at $\alpha = 5 \times 10^{-4}$. We suspect that this is because, in MLP-Mixer, we are simultaneously interpolating three different sources of inductive bias (patchifying, token-mixing, and channel-mixing) as outlined in Section 2. Each inductive bias exhibits a minimum at a different $\alpha$. The final result is the superposition of multiple different V-shapes, with potential constructive and/or destructive interference.

At the endpoints, the I-MLP performance approaches S-MLP at $\alpha = 0$ and the prior model (S-CNN or MLP-Mixer) at $\alpha = 1$.

(*a*) I-MLP (with CNN prior)

| Model | CIFAR-10 | CIFAR-100 |
|---|---|---|
| I-MLP ($\alpha = 0.0$) | 58.36 $\pm$ 0.44 | 29.06 $\pm$ 0.21 |
| I-MLP ($\alpha = 0.01$) | 51.50 $\pm$ 0.86 | 21.67 $\pm$ 1.75 |
| I-MLP ($\alpha = 1.0$) | 62.14 $\pm$ 1.02 | 29.82 $\pm$ 0.99 |
| S-CNN | 62.60 $\pm$ 1.20 | 32.15 $\pm$ 0.57 |

(*b*) I-MLP (with MLP-Mixer prior)

| Model | CIFAR-10 | CIFAR-100 |
|---|---|---|
| I-MLP ($\alpha = 0.0$) | 59.35 $\pm$ 0.22 | 31.00 $\pm$ 0.21 |
| I-MLP ($\alpha = 0.01$) | 52.34 $\pm$ 0.80 | 23.96 $\pm$ 0.22 |
| I-MLP ($\alpha = 1.0$) | 63.51 $\pm$ 0.11 | 33.89 $\pm$ 0.20 |
| MLP-Mixer | 63.42 $\pm$ 0.48 | 33.71 $\pm$ 0.13 |

Table 1: CIFAR-10 and CIFAR-100 test dataset top 1 accuracy scores (mean $\pm$ standard deviation). We include $\alpha = 1.0$ to emphasize the equivalence of I-MLP-CNN with S-CNN and I-MLP-Mixer with MLP-Mixer when $\alpha = 1$. Numerical results confirm this.

### 4.2. Additional Experiments: Test-time Only Interpolation, Interpolation Weight Decay, and First-layer Only Interpolation

We perform three additional experiments with different interpolation strategies. In Appendix F, we compare interpolation during training with test-time only interpolation. We separately train the MLP and prior models, and only use the interpolation weight once before test and inference time. The results in Table 4 show that test-time only interpolation performs significantly worse.

In Appendix G, we experiment with non-constant interpolation weights during training with $\alpha[t]$ as a function of the epoch $t$. We define a decaying interpolation weight schedule of $\alpha[t] = a(1 - \frac{t}{t_{max}})^k$, where $a$ represents the initial interpolation weight and $k$ controls the decay rate. We observe in Figure 3 that interpolation with a constant weight $k = 0$ performs similarly to strong weight decay $k > 4$, with linear decay $k = 1$ performing significantly worse.

In Appendix H, we experiment with more time- and space-efficient interpolation strategies by limiting the number of interpolation operations. We constrain the number of weight matrix parameters being interpolated at each epoch and compare interpolating only a wider first layer versus interpolating multiple narrower layers. For this fixed interpolation parameter budget, we observe that a wide first layer interpolation with a CNN prior is better as shown in Table 5. This empirical finding is important because we can improve training computational efficiency and potentially extend our results to larger datasets than CIFAR-10 and CIFAR-100.

## 5. Summary

In this work, we explore novel training techniques that allow one to continuously increase or decrease the inductive bias present in a plain MLP. Through an interpolation strategy with more structured priors, we study the relationship between MLP performance and fractional inductive bias at low-compute scales. Our experiments strengthen previous results on the role of inductive bias; we believe that our novel training technique can be useful for more controlled analyses in this line of work. For example, in the mid-compute regime, a fractional amount of inductive bias may be the optimal design. Future work could explore more involved interpolation strategies and advanced data augmentation pipelines. Experimenting with larger compute scales and investigating the role of inductive bias in such a regime makes for exciting future work.

## References

[1] Gregor Bachmann, Sotiris Anagnostidis, and Thomas Hofmann. Scaling MLPs: A Tale of Inductive Bias. *Advances in Neural Information Processing Systems*, 36, 2024.

[2] Esko G Cate and David W Twigg. Analysis of in-situ transposition. *ACM Transactions on Mathematical Software (TOMS)*, 3(1):104–110, 1977.

[3] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Jungong Han, and Guiguang Ding. RepMLPNet: Hierarchical Vision MLP with Re-parameterized Locality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 578–587, 2022.

[4] 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.

[5] 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.

[6] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Toronto, Ontario, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[8] 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.

[9] Phillip Wang. MLP Mixer Pytorch. `https://github.com/lucidrains/mlp-mixer-pytorch`, 2022.

## Appendix A. Construction of $\mathbf{W_p}$ from a CNN layer

The conversion of a 2-D convolution kernel $F$ into a fully connected layer $\mathbf{W_F}$ was derived by [3, Section 3]. We adapt the formula here to allow for different input and output dimensions, i.e. when $h_{out} \neq h_{in}$ and $w_{out} \neq w_{in}$:

$$\mathbf{W_F} = \text{reshape}(F * \mathbf{x}^{(I)}, (ch_{in}w_{in}, oh_{out}w_{out}))^T, \tag{6}$$

where

$$\mathbf{x}^{(I)} = \text{reshape}(\mathbf{I}, (ch_{in}w_{in}, c, h_{in}, w_{in})), \tag{7}$$

and $\mathbf{I}$ is the identity matrix with dimensions $(ch_{in}w_{in}, ch_{in}w_{in})$. The function reshape$(\mathbf{x}, dim)$ reshapes a tensor $\mathbf{x}$ into dimension $dim$ in-order.

## Appendix B. Construction of $\mathbf{W_p}$ from MLP-Mixer

For the MLP-Mixer, we require the equivalent fully-connected layer representation $\mathbf{W_P} = \tilde{\mathbf{W}}\mathbf{L}$, where $\mathbf{L} = \mathbf{P}$ or $\mathbf{L} = \mathbf{T}$ for the patchify and transpose operations, and $\tilde{\mathbf{W}}$ corresponds to the expanded Toeplitz matrix for weight sharing. In the following subsections, we show the constructions for $\mathbf{L} = \mathbf{T}$ (Appendix B.1.1), $\mathbf{L} = \mathbf{T}$ (Appendix B.1.2), and $\tilde{\mathbf{W}}$ (Appendix B.2).

### B.1. Linear operation matrix L

#### B.1.1. PATCHIFYING MATRIX P

We first show the construction when $\mathbf{L} = \mathbf{P}$ is the patchifying matrix for tensor $\mathbf{x}$. For notational simplicity, we assume a single channel, i.e. $\mathbf{x} \in \mathbb{R}^{H \times W}$. We denote:

- $(H, W)$: dimensions of the input tensor.
- $(h, w)$: Number of patches along height and width. $h = \frac{H}{P}$ and $w = \frac{W}{P}$.
- $(P, P)$: dimensions of each patch.
- $(R, C)$: patch index (denoting the order of one patch), $R \in \{0, 1, \dots, h-1\}$ and $C \in \{0, 1, \dots, w-1\}$.
- $(k, l)$ : (absolute) pixel index within the tensor $\mathbf{x}$.
- $(r, c)$: (relative) pixel index (denoting the order of pixels within the patch). $r, c \in \{0, 1, \dots, P-1\}$.

The $(k, l)$-th pixel in tensor $\mathbf{x}$ takes the following position in $vec(\mathbf{x})$.

$$idx(k, l) := (k \cdot W + l) \tag{8}$$

For a given patch $(R, C)$ and relative pixel within the patch $(r, c)$, the top-left corner of patch $(R, C)$ in the original image is at:

$$(k_0, l_0) = (R \cdot P, C \cdot P). \tag{9}$$

For each element within the patch, the corresponding index in tensor $\mathbf{x}$ is:

$$(k, l) = (k_0 + r, l_0 + c) \tag{10}$$

Substituting (10) to (8) gives the absolute index in $\mathbf{x}$:

$$idx(r, c) = (k_0 + r) \cdot W + (l_0 + c). \tag{11}$$

Next, we substitute $(k_0, l_0)$ using (9):

$$idx(R, C, r, c) = (R \cdot P + r) \cdot W + (C \cdot P + c). \tag{12}$$

**Construction of P.** Observe that $\mathbf{P}_{ij} = 1$ denotes "$x_j$ should be permuted to position $i$".

For fixed $i$, we first calculate the patch number and $(R, C)$, as well as the relative pixel number and $(r, c)$. The patch number is:

$$\left\lfloor \frac{i}{P^2} \right\rfloor, \tag{13}$$

with patch index:

$$(R, C) = \left( \left\lfloor \left( \frac{i}{P^2} \right) \frac{1}{w} \right\rfloor, \left( \frac{i}{P^2} \right) \bmod w \right) \tag{14}$$

Within the patch $(R, C)$, the pixel has pixel number:

$$i \bmod P^2, \tag{15}$$

and relative pixel index:

$$(r, c) = \left( \left\lfloor \left( i \bmod P^2 \right) \frac{1}{P} \right\rfloor, \left( i \bmod P^2 \right) \bmod P \right). \tag{16}$$

Finally, we use $(R, C)$ and $(r, c)$ and calculate the source index $j$ of input tensor $\mathbf{x}$:

$$\mathbf{P}_{ij} = \begin{cases} 1 & \text{if } j = (R \cdot P + r) \cdot W + (C \cdot P + c), \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

where $(R, C)$ and $(r, c)$ follow from (14) and (16).

### B.1.2. TRANSPOSE MATRIX $\mathbf{T}$

We now show the construction of the transpose matrix $\mathbf{L} = \mathbf{T}$ to transpose a single channel tensor $\mathbf{x} \in \mathbb{R}^{H \times W}$ to $\mathbf{x}^{\mathbf{T}} \in \mathbb{R}^{W \times H}$.

We adapt the in-place transpose permutation algorithm from [2], where $\pi(k)$ indicates the original index into a 2D matrix stored row-wise and $k$ indicates the new index after in-place transposition.

$$\pi(k) = \begin{cases} Wk \bmod HW - 1 & \text{if } k \neq HW - 1, \\ HW - 1 & \text{if } k = HW - 1. \end{cases} \tag{18}$$

Note that $\mathbf{T}_{ij} = 1$ denotes "$x_j$ should be permuted to position $i$". The desired transpose transformation is then:

$$\mathbf{T}_{ij} = \begin{cases} 1 & \text{if } j = \pi(i), \\ 0 & \text{otherwise,} \end{cases} \tag{19}$$

## B.2. Equivalent fully connected layer for weight sharing

Consider an input tensor $\mathbf{X} \in \mathbb{R}^{R \times C}$. We can interpret this input tensor as a set of row vector inputs $\mathbf{x_r} \in \mathbb{R}^C$. The MLP-Mixer, for example, uses a set of row vectors $\mathbf{x_r}$ in its Channel Mixer where each $\mathbf{x_r}$ represents a patch.

We can apply the same MLP layer to each of the row vectors. We denote this as an MLP with shared weights because the same weights are used for each row vector. For simplicity, we express the shared weight matrix formulation using weight matrices where the input and output dimensions are the same. Note, however, that the linear patch embeddings in MLP-Mixer do not use isotropic weight matrices.

The shared weight $\mathbf{w_r} \in \mathbb{R}^{C\times C}$ acts on each row of the input tensor $\mathbf{X} \in \mathbb{R}^{R\times C}$ and produces an output tensor $\mathbf{Y} \in \mathbb{R}^{R\times C}$. For the $i$-th row,

$$\mathbf{y_i} = \mathbf{x_i}\mathbf{w_r}^T, \quad i = 1, \cdots, R \tag{20}$$

where $x_i \in \mathbb{R}^C$ and $y_i \in \mathbb{R}^C$ are the $i$-th row of $\mathbf{X}$ and $\mathbf{Y}$ respectively. Equation (20) can be alternatively expressed as a Toeplitz matrix $\mathbf{W} \in \mathbb{R}^{RC\times RC}$ acting on the vectorized form of $\mathbf{X}$ and $\mathbf{Y}$:

$$vec(\mathbf{Y}) = vec(\mathbf{X})\mathbf{W}^T, \tag{21}$$

where the shared linear layer layer $\mathbf{w}$ is repeated on the diagonal of $\mathbf{W}$.

$$[\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_R}] = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_R}] \begin{bmatrix} \mathbf{w_r} & 0 & \cdots & 0 \\ 0 & \mathbf{w_r} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{w_r} \end{bmatrix}^T \tag{22}$$

Therefore, the explicit weight matrix is $\mathbf{W} = diag(\mathbf{w_r})$.

## Appendix C. I-MLP Architecture Details - CNN

We design an Interpolated MLP (I-MLP) that interpolates between the S-MLP and S-CNN. We summarize the S-MLP, I-MLP, and S-CNN architectures in Table 2 for batch size $n$ and image shape $32 \times 32 \times 3$. To allow for interpolation, we require that in each layer, the dimensions are consistent across all three models, i.e. $d_{MLP} = d_{CNN}$. Since CNNs require multiple channels to be performant, we increase the output channel $o$ of each layer in an encoder-like fashion. To counteract the increase in channel dimensions, we decrease $h_{out}, w_{out}$ by using a kernel size $k \times k = 3 \times 3$, stride $s = 2$, and padding $p = 1$ to satisfy the $d_{CNN} = d_{MLP} = 1024$ constraint. The layer output height $h_{out}$ and width $w_{out}$ are governed by $h_{out} = \left(\frac{h_{in}-k+2p}{s}\right) + 1$ and $w_{out} = \left(\frac{w_{in}-k+2p}{s}\right) + 1$. For all three architectures, the first layer serves as an embedding layer to convert images with 3 RGB channels to 1 channel and the final layer is a linear classifier. Following [1], we use Layer Norm for normalization and GELU as the activation function.

## Appendix D. I-MLP Architecture Details - MLP-Mixer

We design an Interpolated MLP (I-MLP) that interpolates between the S-MLP and the MLP-Mixer. We summarize the S-MLP, I-MLP, and MLP-Mixer architectures in Table 3 for batch size $n$ and image shape $32 \times 32 \times 3$.

The MLP-Mixer consists of repeated groups of layers called Mixer Layers, with the MLP-Mixer depth indicated the number of Mixer Layers. Each Mixer Layer operates on a 2D tensor $\mathbf{x} \in \mathbb{R}^{S\times C}$, where $S$ indicates the number of patches and $C$ indicates the number of channels after performing linear patch embedding. Within each Mixer Layer are two 2-layer MLPs with shared weights called the Token Mixer and the Channel Mixer. First, the Token Mixer acts on the patch dimension; it

| MLP Layer | CNN Layer | MLP Output Shape | CNN Output Shape |
|---|---|---|---|
| **Layer-1** | Convolution 2D | [n, 1024] | [n, 1, 32, 32] |
| **Layer-2** | Convolution 2D | [n, 1024] | [n, 4, 16, 16] |
| **Layer-3** | Convolution 2D | [n, 1024] | [n, 16, 8, 8] |
| **Layer-4** | Convolution 2D | [n, 1024] | [n, 64, 4, 4] |
| **Layer-5** | Convolution 2D | [n, 1024] | [n, 256, 2, 2] |
| **Layer-6** | Convolution 2D | [n, 1024] | [n, 256, 2, 2] |
| Linear-7 | Classification Head | [n, 10] | [n, 10] |

Table 2: Model architectures for MLP (S-MLP and I-MLP) and CNN (S-CNN) networks used. The fully connected and convolutional layers used for interpolation are bolded. For both models, the features from Layers-1 and Layers-5 are normalized (through a layer norm) then activated using GELU. The S-MLP and I-MLP models have 8,405,002 parameters, while the S-CNN model uses 757,982 parameters.

applies its MLP layers with shared weights to each column vector $\mathbf{x_c} \in \mathbb{R}^S$. Then, the Channel Mixer acts on the channel dimension: it applies its MLP layers with shared weights to each row vector $\mathbf{x_r} \in \mathbb{R}^C$. To implement this behavior, we apply a transpose linear transformation $\mathbf{L} = \mathbf{T}$ to the input and output of the Token Mixer. This approach allows us to use the same matrix $\tilde{\mathbf{W}}$ formulation for weight sharing along rows.

As with the I-MLP with CNN prior, we require for interpolation that in each layer, the dimensions are consistent across all three models, i.e. $d_{MLP} = d_{Mixer}$. We use an $8 \times 8$ patch size, $S = \frac{HW}{p^2} = 16$ patches, $C = 128$ hidden channels, and an MLP-Mixer depth of 2. Therefore, $d_{MLP} = d_{Mixer} = SC = 2048$.

For all three architectures, the first layer serves as an embedding layer to convert images with 3 RGB channels to $C$ channels and the final layer is a linear classifier. Following [1], we use Layer Norm for normalization and GELU as the activation function.

## Appendix E. Experiment setups

**Setup.** We build on open-sourced PyTorch implementations of S-MLP [1], RepMLPNet [3], and MLP-Mixer [8][9] and train using the NVIDIA GTX Titan X. For evaluation, we use $32 \times 32 \times 3$ RGB images from the CIFAR-10 [6] and CIFAR-100 [6] datasets. We train for 100 epochs with the Adam optimizer using Cross Entropy Loss, a learning rate of 0.0001, and a batch size of 128. We use S-MLP as our baseline and report the top 1 accuracy score on the test dataset.

**Dataset Augmentation.** Similar to Bachmann et al., we note that data augmentation is very important for the S-MLP and I-MLP. When training for 100 epochs without data augmentation, the S-MLP, I-MLP (for all values of $\alpha$), and CNN all show signs of overfitting with the validation cross entropy loss starting to increase around 10 epochs. We also note that the I-MLP with $\alpha = 0.5$, I-MLP with $\alpha = 1.0$ and CNN perform worse than the S-MLP without data augmentation due to

| MLP Layer | MLP-Mixer Layer | MLP Output Shape | MLP-Mixer Output Shape |
|---|---|---|---|
| **Layer-1** | Linear Patch Embedding | [n, 2048] | [n, 16, 128] |
| **Layer-2** | Token Mixer MLP 1† | [n, 2048] | [n, 128, 16] |
| **Layer-3** | Token Mixer MLP 2†† | [n, 2048] | [n, 16, 128] |
| **Layer-4** | Channel Mixer MLP 1 | [n, 2048] | [n, 16, 128] |
| **Layer-5** | Channel Mixer MLP 2 | [n, 2048] | [n, 16, 128] |
| **Layer-6** | Token Mixer MLP 1† | [n, 2048] | [n, 128, 16] |
| **Layer-7** | Token Mixer MLP 2†† | [n, 2048] | [n, 16, 128] |
| **Layer-8** | Channel Mixer MLP 1 | [n, 2048] | [n, 16, 128] |
| **Layer-9** | Channel Mixer MLP 2 | [n, 2048] | [n, 16, 128] |
| Linear-10 | Classification Head | [n, 10] | [n, 10] |

Table 3: Model architectures for MLP (S-MLP and I-MLP) and MLP-Mixer networks used. The fully connected layers with and without weight sharing used for interpolation are bolded. There are two separate Mixer Layers: Layer-2 to Layer-5 and Layer-6 to Layer-9. For both models, the features from the Linear Patch Embedding (Layer-1), the Token Mixer MLP 2 (Layer-3 and Layer 7), Channel Mixer MLP 2 (Layer-5 and Layer-9) are normalized (through a layer norm) while the features between the 2 layers of each Token Mixer (Layer-2, Layer-6) and Channel Mixer (Layer-4, Layer-8) are activated using GELU. The S-MLP and I-MLP models have 39,865,610 parameters, while the MLP-Mixer model uses 93,130 parameters.
†indicates that the layer input is transposed
††indicates that the output layer is transposed

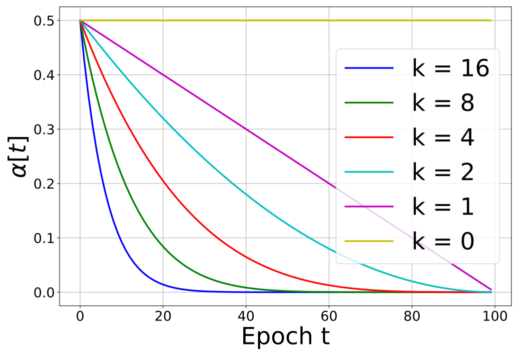
this overfitting issue. After applying the same data augmentation transforms as Bachmann et al. by using random crops and horizontal flips, we see that accuracy improves for all models and the I-MLP with $\alpha = 0.5$ and $\alpha = 1.0$ both significantly improve and outperform the S-MLP.

## Appendix F. Test-time only interpolation

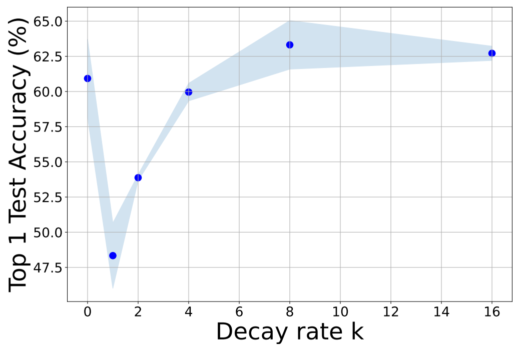
To validate the correctness of our interpolation experiment, we separately train the MLP and prior models and only use the interpolation weight at test and inference time. This is equivalent to no interpolation during training, and then interpolating the MLP and prior model once before inference with $\alpha_{\text{test}}$. We expect this to perform worse than using the MLP or prior model separately, since the two models will have converged differently. Our experiments with a CNN prior model shown in Table 4 confirm that the interpolation at test time performs worse.

## Appendix G. Interpolation weight decay: Varying interpolated bias over time

We experimented with non-constant interpolation weights during training with $\alpha[t]$ as a function of the epoch $t$. We define a decaying interpolation weight schedule of $\alpha[t] = a(1 - \frac{t}{t_{max}})^k$, where $a$ represents the initial interpolation weight and $k$ controls the rate of decay. Note that with our original constant interpolation weight $\alpha[t] = a$, the inductive bias added from the prior model forms a geometric series. To counteract this compounding effect, we use the decaying weight schedule. We vary the decay rate $k$ for a fixed initial interpolation weight $a = 0.5$ in Figure 3 and Table 4. We observe that constant interpolation with $k = 0$ performs similarly to a high decay rate $k > 4$, with a sharp local minimum at $k = 1$ for linear interpolation weight decay. We suspect that changing the interpolation weight $\alpha[t]$ leads to each weight update moving in opposite directions and countering the previous weight updates. With a high decay rate, the interpolation weight $\alpha[t]$ quickly decays to $\alpha[t] = 0$ and the weight updates converge in the same direction.



(*a*) $\alpha$ decay schedule



(*b*) Test accuracy versus $\alpha$ decay rate $k$

Figure 3: CIfAR-10 test accuracy of I-MLP when interpolated with a CNN using varying decay rate $k$ and initial interpolation weight $a = 0.5$.

| Model | CIFAR-10 |
|---|---|
| CNN | $62.60 \pm 1.20$ |
| I-MLP no interpolation $\alpha[t] = 0$ | $58.36 \pm 0.44$ |
| I-MLP constant interpolation $\alpha[t] = 0.5$ | $60.98 \pm 1.97$ |
| I-MLP test-time interpolation $\alpha_{\text{test}} = 0.5$ | $48.63 \pm 1.68$ |
| I-MLP linear decay interpolation $a = 0.5, k = 1$ | $48.34 \pm 2.31$ |
| I-MLP exponential decay interpolation $a = 0.5, k = 2$ | $53.88 \pm 0.21$ |
| I-MLP exponential decay interpolation $a = 0.5, k = 4$ | $59.96 \pm 0.62$ |

Table 4: CIFAR-10 test accuracy scores for a standard MLP, standard CNN, and I-MLP with various types of non-constant inductive bias interpolation methods.

## Appendix H. I-MLP in practice: the first layer matters the most

We created two separate MLPs (MLP-1 and MLP-2) with a different shape but a similar number of total weight matrix parameters. Both MLPs are interpolated with a CNN prior model; however, we constrain interpolation to a fixed budget of 944,0256 parameters. In MLP-1, the interpolation parameters are spread across 6 layers; 6 layers are interpolated. In MLP-2, the interpolation parameters are concentrated in the first layer, which is achieved by making the first layer in MLP-2 wider.

To isolate the performance difference to only the interpolation of the first layer versus multiple layers, we confirm that our two MLPs have the a similar number of total parameters and that they perform similarly without interpolation (see baseline in Table 5).

We observe that with all interpolation parameters concentrated in the first layer, MLP-2 leverages the inductive bias from CNN-interpolation much more effectively. The CIFAR-10 top 1 accuracy improved from 54.8% to 73.7% in MLP-2 (v.s. 56.4% to 60.3% in MLP-1). Similar improvements in CIFAR-100 are also observed.

| | | | CIFAR-10 | | CIFAR-100 | |
|---|---|---|---|---|---|---|
| Model | # total params | # interpolated params | baseline | CNN-interpolated | baseline | CNN-interpolated |
| MLP-1 | 16,922,724 | 944,0256 | 56.4% | 60.3% | 25.6% | 27.3% |
| MLP-2 | 16,812,024 | 944,0256 | 54.8% | 73.7% | 24.4% | 40.6% |

Table 5: Parameters and performance comparison of a constant-width (MLP-1) and first-layer-concentrated MLP (MLP-2). Baseline denotes training with no interpolation. CNN-interpolated denotes training with interpolation from a CNN. We show that with a fixed number of interpolated parameters (hence fixed interpolation compute), it is best to concentrate the interpolation parameters in the first layer to capture the most inductive bias.