Adapting Multi-Input Multi-Output schemes to Vision Transformers

Rémy Sun[1,3,*] Clément Masson[3] Nicolas Thome[2] Matthieu Cord[1,4]

[1]MLIA, ISIR, Sorbonne Université

[2]Vertigo, CEDRIC, Conservatoire National des Arts et Métiers
[3]Thales Land and Air Systems, Elancourt, France [4]Valeo.ai

Abstract

Multi-input multi-output models have proven capable of providing ensembling for free in convolutional neural networks by training independent subnetworks that can be ensembled. How these architectures translate to vision transformer architectures is however unclear. In this paper, we introduce MixViT, the first multi-input multi-output framework for vision transformers. After discussing MixViT and the novel source embedding mechanism it relies on for subnetwork separation, we show it significantly improves on standard transformers in a preliminary study on CIFAR100.

1. Introduction

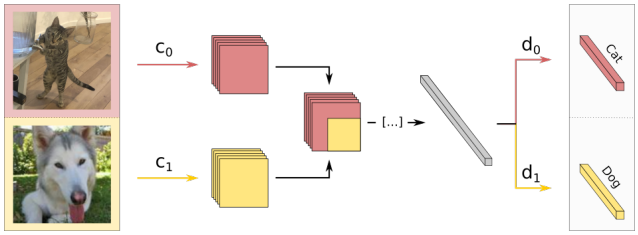
Deep architectures have cemented themselves as the reference in many application domains [6, 8] over the last decade. Finding ways to maximize model performance has therefore become a particularly fruitful pursuit [15, 20].

Ensembling the predictions of multiple independently trained models [9] has long been established to significantly improve model performance. Indeed, it has been shown the predictions from the different model can be aggregated to yield much more reliable predictions.

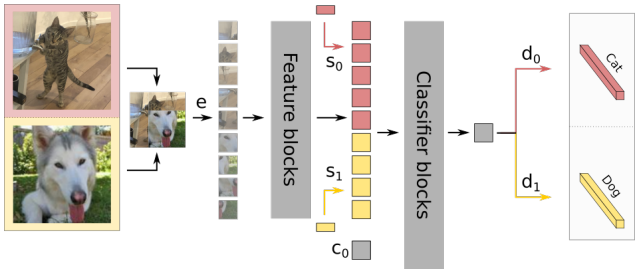
However, ensembling also incurs significant overhead costs as the models must be trained in the first place [4, 9]. As such, minimizing the overhead induced by ensembling - e.g. through some measure of parameter sharing - is of paramount importance in the ensembling literature [10, 18].

Multi-input multi-output (MIMO) strategies [5, 14] constitute a very interesting solution to this problem: ensembling in a MIMO framework is virtually free (see Sec. 2 and Fig. 1a). Indeed, MIMO models take advantage of large neural networks' sparse nature to train [12] independent subnetworks within a base model by defining multiple input/output pairs on the core network.

*Corresponding author mailto: remy.sun@isir.upmc.fr



(a) MIMO in CNNs.



(b) Our proposed MIMO transformer, MixViT.

Figure 1. We propose MixViT a multi-input multi-output framework for vision transformers that displays significant performance gains in practice (e.g. 78.4% accuracy on CIFAR100 for a standard model vs. 82.4% accuracy for MixViT).

While MIMO have proven very useful in convolutional architectures, these strategies have yet to be successfully deployed in vision transformer models [3, 17]. In this paper, we investigate how MIMO strategies adapt to transformer architectures. We propose MixViT, the first multi-input multi-output framework for vision transformers that outperforms its single input backbone at a reasonable cost. As shown in Fig. 1b, MixVit mixes tokens from M inputs before feeding the mix of tokens to feature extraction blocks. We then introduce a novel "source embedding" that identifies which input each token comes from, and extract features with a class token. Finally, we get M predictions from the class token features with M different dense layers. Interestingly, our new "source embedding" mechanism leverages peculiarities of the transformer architecture to advan-

tageously replace standard CNN based MIMO components.

We start this paper by recalling the CNN-based multi-input multi-output strategy in Sec. 2. In Sec. 3, we detail our MIMO transformer framework MixViT and provide preliminary results on CIFAR100 (Sec. 4) using a variant of the ConViT [2] architecture as a backbone.

2. MIMO structures in CNNs

We provide here a quick refresher on multi-input multi-output architectures in convolutional neural networks, as identifying the key components of the framework is very beneficial in understanding how we adapt MIMO strategies to vision transformers in Sec. 3. For practical reasons, we directly introduce the generalized multi-input multi-output strategy introduced in MixMo [14]. For simplicity, we will discuss the case where $M = 2$ subnetworks are trained unless specified otherwise.

As figured by Fig. 1a for $M = 2$, the M inputs are processed by M subnetworks. Therefore, we consider M (inputs, labels) pairs $\{(x_i, y_i)\}_{0 \leq i < M}$ during training. Plainly speaking, **we feed M inputs to the model at once**. These M inputs are embedded by M different convolutional layers $\{c_i\}_{0 \leq i < M}$ before being mixed into one common representation (either through sums [5] or other mixing schemes [14]). The core network then processes this representation into a common feature vector. M dense layers $\{d_i\}_{0 \leq i < M}$ then yield M predictions from this vector. At test time, M predictions can be obtained for one image by inputting this same image to each subnetwork/encoder.

The difficulty in training multi-input multi-output transformers seems to lie in how the multi-input block translates to vision transformer. This appears to stem from a difference in how the **encoding** and **mixing** components of the framework interact in vision transformers. In CNNs, the $M = 2$ inputs are **encoded** by two different convolutional layers into a shared representation space. This encoding step plays an important role as it allows the model to differentiate information relevant to the multiple subnetworks [14, 16].

The encoded inputs are then - independently of the encoding step - **mixed** into a common representation $h = Mix(c_0(x_0), c_1(x_1), \lambda)$ with a random ratio $\lambda \in [0, 1]$ by a mixing function Mix. As can be inferred from the previous equation, Mix typically draws from the mixing samples data augmentation literature [19, 20]. While the seminal MIMO paper relies on an interpolating mixing operation à la MixUp [20], it has since been shown binary mixing operations that keeps pixels from one input or the other like CutMix [19] are preferable. This is because the mixing operation's fundamental role in MIMO models is to create a compressed representation that contains as much actionable information on the inputs as possible.

Contrarily to CNN-based MIMO frameworks, our proposed MixViT framework separates these two components of the multi-input block.

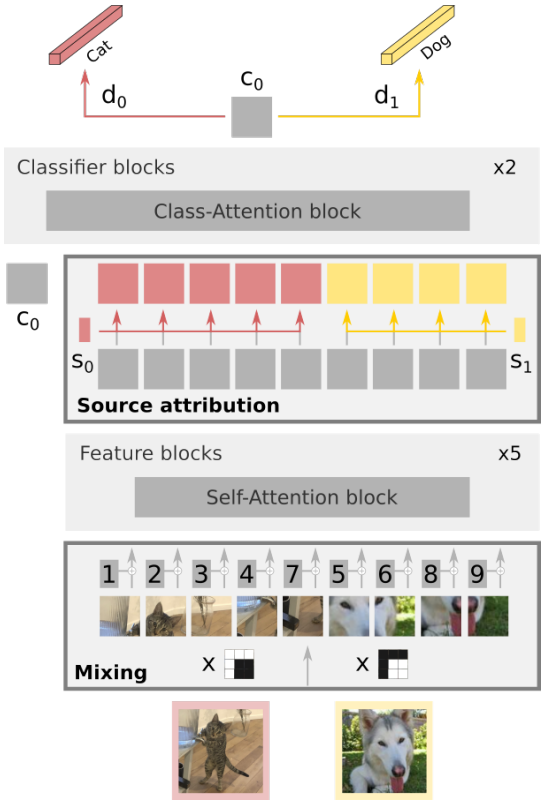


Figure 2. Our proposed MixViT framework on a ConViT backbone. Patch tokens are selected according to a MixToken scheme. A source embedding s_i is added to each subnetwork-agnostic patch token depending on which input the patch is from. M predictions are then extracted from the attended class token features.

3. MixViT

We propose here MixViT, a multi-input multi-output framework suited to vision transformers that relies on a novel "source embedding" to encode the inputs to the different subnetworks. For simplicity, we illustrate MixViT on a ConViT [2] backbone but MixViT can be adapted to most vision transformer architectures.

During training, MixViT - as figured on Fig. 2 - takes as input a mix of N patch tokens from the M inputs (according to a CutMix [19] or rather MixToken [7] scheme), adds a positional embedding to the tokens and feeds them to 5 gated positional self-attention (GPSA) blocks [2]. The resulting mixed set of attended tokens is then marked with **source embeddings** s_i giving the patch's source image and a new classification c_0 token is added to the set of tokens. The tokens are then passed to 2 more self-attention (SA) blocks and the feature representation of the class token is retrieved. M predictions are then obtained from this feature

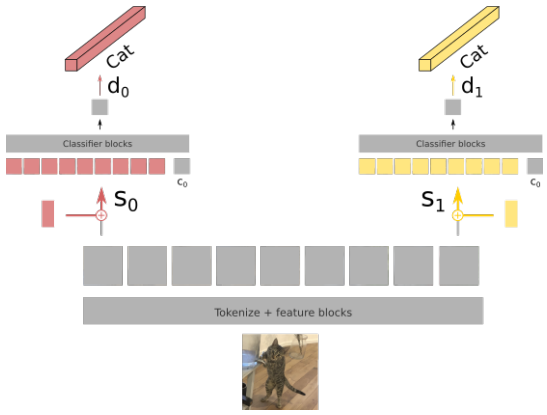


Figure 3. At inference, MixViT extracts subnetwork-agnostic tokens for the whole image. M sets of subnetwork-specific tokens are then passed to the end of the network.

representation through the use of M different dense layers.

At test time, MixViT simply takes as input the tokens of the considered sample and feeds them to the GPSA blocks. We then create M sets of tokens to feed to the rest of the network by duplicating the attended tokens and combining them with one subnetwork's source embedding (see Fig. 3). For each set, we feed the tokens along with the common classification token to the last 2 self-attention blocks and extract the relevant prediction. While this requires M passes through the end of the model, the computational cost incurred remains reasonable ($\times \sim 1.25$ instead of $\times \sim 2.15$).

The crucial difference between MixViT and traditional CNN based MIMO models lies in how the **encoding** step differentiates the subnetworks with source embeddings s_i. First, we directly tie the encoding step to the patch tokens and the **mixing** step to take advantage of vision transformers' internal structure. This idea of leveraging knowledge on the mixing step to guide subnetwork separation has in fact seen limited success to allow subnetworks to share early features in CNNs through the "unmixing" strategy [16]. Unmixing however proves unwieldy in CNNs and often leads to identical subnetworks whereas source embedding provides a natural and simple solution in vision transformers. Second, source embeddings allow us to differentiate subnetworks late in the network and therefore pass uncompressed inputs through the end of the model. Encoding early actually forces CNNs to use compressed inputs which proves problematic in CutMix based MixMo models [14]. These models circumvent the issue by falling back on interpolating mixing at test time and towards the end of training, but this only superficially addresses the issue.

4. Experimental discussion

We first validate our proposed MixViT architecture against some baselines in Sec. 4.1. We then discuss in Secs. 4.2-4.3 the possible implementation choices along with their impact, and how the moving parts of our proposed MixViT relate to the key components of MIMO frameworks in CNNs. Finally, we propose a brief discussion on MIMO methods as mixing data augmentations (Sec. 4.4).

Model	MSDA	Accuracy (%)	Ind. Acc. (%)	Diversity
ConViT [2]		69.4 ± 0.3	69.4 ± 0.3	N/A
MIMO [5] ViT		73.0 ± 0.2	69.8 ± 0.2	0.68 ± 0.03
MixMo [14] ViT		72.4 ± 0.3	69.2 ± 0.1	0.67 ± 0.03
MixViT (ours)		80.7 ± 0.3	79.9 ± 0.1	0.37 ± 0.02
ConViT [2]		78.1 ± 0.3	78.1 ± 0.3	N/A
MIMO [5] ViT	✓	77.2 ± 0.4	74.8 ± 0.4	0.69 ± 0.02
MixMo [14] ViT	✓	76.2 ± 0.3	73.7 ± 0.3	0.69 ± 0.02
MixViT (ours)	✓	82.1 ± 0.2	81.6 ± 0.3	0.35 ± 0.06

Table 1. MixVit improves on both naive MIMO implementations and standard single input single output ConViTs. In particular, MixViT still shows strong performances without mixing samples data augmentation (MSDA: MixUp + CutMix) on inputs.

In this paper, we run preliminary experiments on CIFAR 100 with a variant of the ConViT [2] architecture with 5 GPSA blocks and 2 SA blocks with an embedding dimension of 384 and 12 heads (see Appendix). Our proposed MIMO scheme should however be easily transposed to most common vision transformer architectures. For each result, we report the overall ensemble accuracy, the accuracy of individual subnetworks and diversity (ratio-error) between subnetworks for one seeded run in Tab. 2, and in the form $mean \pm std$ over three seeded runs in Tab. 1.

4.1. Main result

Tab. 1 shows our MixViT method provides significant improvements over a standard ConViT network. Interestingly, while MixViT shows solid gains from ensembling, a large parts of the gains are due to a much higher accuracy of the individual subnetworks: the framework also has a strong regularizing effect, possibly due to the feature extractor being shared. This is further confirmed by the fact ConViT strongly suffers without mixing data augmentations (MixUp + CutMix) on its input [17] whereas MixViT retains good performance with standard inputs. We also compare against one-to-one adaptations of the seminal MIMO [5] and MixMo [14] frameworks to transformers (see Appendix). This second baseline simply uses two different initial encoders for patch tokens and mixes according to a summing or CutMix scheme and duplicating the final dense layer (as is done in MixViT). In accordance with results reported in [1], we verify that this second baseline fails to produce improvements where MixVit succeeds.

4.2. Source embeddings and attention

The source embeddings as an **encoding** mechanism therefore constitute a key contribution of this work. As can

Model	Accuracy (%)	Ind. acc. (%)	Diversity
MixViT (MV)	82.4	81.9	0.30
MV - source embedding	80.8	80.7	0.05
MV - test tiling	81.1	80.5	0.40
MV + class attention	81.9	81.5	0.33
MV + multiple class tokens	78.8	72.5	1.12
MV + all tokens (AT)	80.8	80.2	0.33
MV + AT + input source embedding	80.7	80.3	0.23
MV + AT + different patch encoders	79.8	77.5	0.70

Table 2. Ablation on components and implementation choices in the MixViT framework.

be observed from Tab. 2, this **source embedding** is essential to make the system work. Indeed, the subnetworks otherwise collapse to the same function.

We are able to perform M independent passes at inference (linear cost) instead of one single pass over a concatenation of the M sets of attended tokens (quadratic cost) because the attention mechanism separates patches from different inputs. Indeed, a quick visualization of attention weights (see Appendix) in the last layers of MixViT shows patches attend exclusively to other patches with the same source embedding and mostly ignore patches marked by other source embeddings. This is important as allowing the classifier blocks to see the whole inputs significantly improves performance (see **test-tiling** line in Tab. 2). Incidentally, this means very little is lost when using **class-attention** instead of self-attention towards the end of the network (see Tab. 2) for even less overhead.

Interestingly, this interaction between attention and source embeddings is likely also to blame for the model's inability to properly accommodate **multiple class tokens** (one for each subnetwork). While it seems natural that using one class token for each subnetwork would lead to learning strong independent subnetworks, Tab. 2 shows this is not the case. Note that the average accuracy is deceptive in this case, as only one subnetwork performs well in the reported run. In other preliminary experiments, using multiple tokens actually most often leads to identical subnetworks. This might be because each class token only attends to patches from the relevant input, which is similar to a parallel evaluation of samples in a batch. Worse, these computations use the exact same parameters which makes it difficult to learn to good and different functions. Since the subnetworks do not have to share the same class token, there is no need to learn representations representing both inputs well and the learned functions tend to collapse.

4.3. Enforcing input mixing

The way inputs are **mixed** in our framework mostly serves to compress multiple inputs to make computations efficient and to ensure source embeddings can properly separate inputs later in the network. By selecting patches from the $M = 2$ inputs according to a MixToken [7] scheme (ie, a CutMix [19] scheme with borders that coincide with patch lines), we ensure each patch token representation encode information about a single input. This would not be the case with an interpolating scheme like MixUp.

Compressing inputs might lead to a degradation of our models, but this is not the case. We verify this experimentally by passing **all** $2N$ **tokens** of both inputs instead of N tokens to the network (Tab. 2), though this is a wildly unrealistic setting. Our compressed scheme on the other hand keeps computational costs reasonable while exhibiting even better performance.

MixViT only performs subnetwork **encoding** long after the **mixing** step whereas the two steps are often performed together. To validate our choice, we tested performing the encoding and mixing steps simultaneously. Since it is very important to keep each patch token "pure", we would need to create M subnetwork-specific sets of tokens from the start at test time: this would be very inefficient in any case. To simplify comparisons, we instead reprise the previous concatenated inputs setting ("AT") and move the **encoding** step to the beginning. Tab. 2 shows **moving the source embeddings** slightly degrades performance. Similarly poor results are obtained if we use **two different linear layers** in traditional MIMO fashion to embed the patch tokens at the beginning instead of adding a source embeddings. Interestingly however, using different patch embeddings does seem to lead to more diverse predictions.

4.4. MIMO frameworks as data augmentations

MIMO CNNs can be understood as an in-manifold mixing samples data augmentation (MSDA) [14]. Indeed, MIMO schemes mix two inputs x_0 and x_1 in the first network manifold with ratio λ before (roughly) optimizing a mixed loss $\lambda l_0 + 1 - \lambda l_1$. As such, the main difference lies in the fact that MSDA frameworks use the same classifier for both inputs ($d_0 = d_1$).

This is even more pronounced in MixViT as mixing is performed directly in patch/pixel space on a standard MixToken [7] input. MixViT's feature extractor is wholly shared between subnetworks and seems to benefit from this (see Tab. 2). Furthermore, much of MixViT's improvements on the single input ConViT backbone must be credited to the model having very strong individual subnetworks due to an inherent regularization. It is therefore perhaps not surprising that MixViT in itself provides sufficient regularization to train ConViT networks even without applying CutMix and MixUp on the inputs (see Tab. 1).

5. Conclusion

We have proposed MixViT, the first multi-input multi-output framework for vision transformers that showcases

significant improvements over the base model for little additional cost. MixViT introduces a novel separation of multi-input block components and separates the subnetworks through a new source embedding. Interestingly, vision transformers prove much more prone to train subnetworks that share features and benefit from co-training. We hope this work leads to further exploration of transformer-based MIMO frameworks, especially as recent work [13] suggests transformers can accommodate a much larger number of subnetworks compared to CNNs.

Acknowledgments This work was conducted using HPC resources from GENCI–IDRIS (Grant 2021-AD011013208), and under a CIFRE grant between Thales Land and Air Systems and Sorbonne University.

References

[1] James Urquhart Allingham, Florian Wenzel, Zelda E Mariet, Basil Mustafa, Joan Puigcerver, Neil Houlsby, Ghassen Jerfel, Vincent Fortuin, Balaji Lakshminarayanan, Jasper Snoek, Dustin Tran, Carlos Riquelme Ruiz, and Rodolphe Jenatton. Sparse moes meet efficient ensembles. *Arxiv preprint*, 2021. 3

[2] Stéphane D'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, 2021. 2, 3, 6

[3] 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. 1

[4] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 1990. 1

[5] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *International Conference on Learning Representations*, 2021. 1, 2, 3, 6

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016. 1

[7] Zi-Hang Jiang, Qibin Hou, Li Yuan, Daquan Zhou, Yujun Shi, Xiaojie Jin, Anran Wang, and Jiashi Feng. All tokens matter: Token labeling for training better vision transformers. In *Advances in Neural Information Processing Systems*, 2021. 2, 4

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 1

[9] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, 2017. 1

[10] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *Arxiv preprint*, 2015. 1

[11] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *International Conference on Computer Vision (ICCV)*, 2021.

[12] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, 2020. 1

[13] Vishvak Murahari, Carlos E. Jimenez, Runzhe Yang, and Karthik Narasimhan. Datamux: Data multiplexing for neural networks. *Arxiv preprint*, 2022. 5

[14] Alexandre Rame, Remy Sun, and Matthieu Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *International Conference on Computer Vision*, 2021. 1, 2, 3, 4, 6

[15] Divya Shanmugam, Davis Blalock, Guha Balakrishnan, and John Guttag. Better aggregation in test-time augmentation. In *International Conference on Computer Vision*, 2021. 1

[16] Remy Sun, Alexandre Rame, Clement Masson, Nicolas Thome, and Matthieu Cord. Towards efficient feature sharing in mimo architectures. *Arxiv preprint*, 2022. 2, 3

[17] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers and distillation through attention. In *International Conference on Machine Learning*, 2021. 1, 3

[18] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2019. 1

[19] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision*, 2019. 2, 4

[20] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 1, 2

Appendix

A. Experimental details

The code for this work was directly adapted from the official MixMo [14] codebase: https://github.com/alexrame/mixmo-pytorch. For contractual reasons, the exact codebase used for this work cannot be released as of yet.

We used a variant of the ConViT [2] architecture, with patch size 4×4, hidden dimension 384, 12 attention heads, 5 GPSA blocks and 2 SA Blocks. We re-use the **hyperparameters** configuration from MIMO [5] with batch repetition 2 (bar2), trained over 500 epochs . The optimizer is Adam with learning rate of 0.0002, batch size 64, linear warmup over 1 epoch, decay rate 0.1 at steps $\{400, 450\}$, l_2 regularization 5e-5. Our experiments mostly ran on a single NVIDIA 12Go-TITAN X Pascal GPU.

All experiments in Tab. 1 were run three times on three fixed seeds from the same version of the codebase. Quantitative results presented in Tab. 1 are given in the form of $mean \pm std$ over the three runs. Only one run on the first seed is reported in Tab. 2.

MIMO and MixMo baselines It is technically perfectly possible to implement MixMo and MIMO in vision transformers, though it does not lead to performance gains in practice. There are only two adaptations that must be made on a standard ConViT network in a fashion completely analogous with CNN-based MIMO frameworks (see Fig. 1a). First, the linear patch embedding layer that maps patches of pixel values to 384-dimensional embeddings must be duplicated, such that we have one embedding layer per input. After encoding the inputs with the embedding layers, we mix the tokens following a mixing scheme. In the MIMO case we use a sum. In the MixMo case we use a CutMix scheme with probability 0.5 and MixUp with probability 0.5. In both cases, we always perform a sum at test time. Secondly, we duplicate the final dense layer used for classification: given the final feature distribution extracted by the class token we apply two different dense layers to get two predictions.

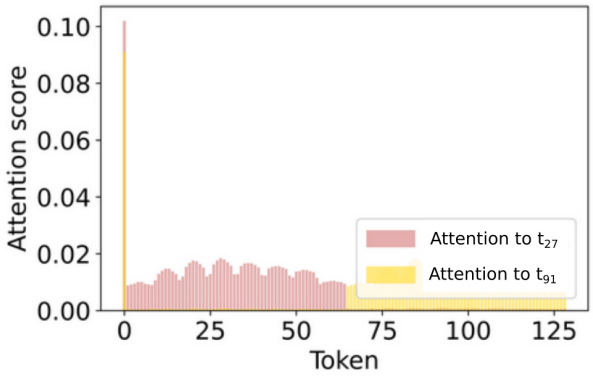


Figure 4. Attention of tokens 27 (center of input 0) and 91 (center of input 1) to other tokens in the first SA block of the classifier blocks. Token 0 is the classification token.

B. Attention in the classifier block

We visualize the impact of the source embedding on the attention mechanism of the first self-attention block in the classifier block in Fig. 4 in the "all token" case. The figure shows the attention of the central tokens of both inputs with respect to the other tokens. As can be observed, these patch tokens are only put in relation with other tokens from the same input and the classifier token: they effectively ignore tokens from the other input.