MLP-Mixer: An all-MLP Architecture for Vision

- [[📚 2105.01601v4.pdf]]
- **Basic Idea**:
  - Can we construct networks for Vision without using CNNs or self-attention (ViTs)?
  - Can we explicitly segregate "mixing" the per-location features (doing operations on image patches) and "mixing" spatial information (doing operations across image patches)?
  - Mixer-L/16 (for input resolution 224×224 and excluding classifier head): 207 million parameters, ImageNet accuracy: 84.15% (top1)

- **Mixer Architecture**:
  - 🟡 **P2**

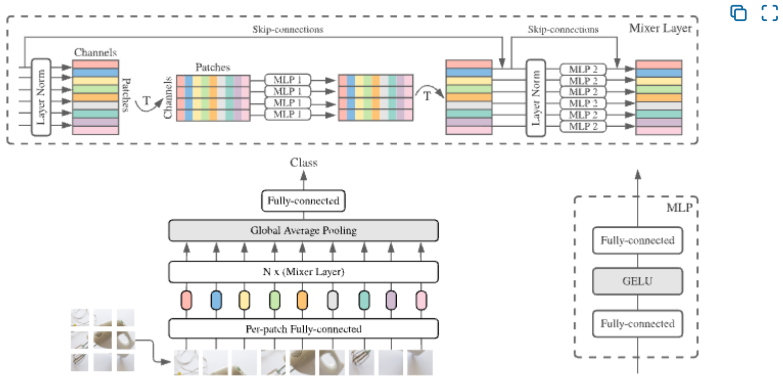

Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.

  - *Generating Patches* is similar to ViT.
    Number of Patches S $\implies$ transform/project each patch using the same projection matrix to hidden dimension $\implies$ Output: Two-dimensional real-valued input table $X \in \mathbb{R}^{S \times C}$.
  - MLP mixer consists of multiple layers: each layer consists of two MLP blocks.
    - *Token Mixing MLP* ($\mathbb{R}^S \mapsto \mathbb{R}^S$): Acts on the columns of X and is shared across columns.
    - *Channel Mixing MLP* ($\mathbb{R}^C \mapsto \mathbb{R}^C$): Acts on the rows of X and is shared across rows.
  - 🟡 **P2** Each MLP block contains two 🟡 **P3** fully-connected layers and a nonlinearity applied independently to each row of its input data tensor. Mixer layers can be written as follows (omitting layer indices):

    🟡 **P3**

$$
\begin{aligned}
\mathbf{U}_{*,i} &= \mathbf{X}_{*,i} + \mathbf{W}_2\,\sigma\big(\mathbf{W}_1\,\mathrm{LayerNorm}(\mathbf{X})_{*,i}\big), && \text{for } i = 1 \ldots C,\\
\mathbf{Y}_{j,*} &= \mathbf{U}_{j,*} + \mathbf{W}_4\,\sigma\big(\mathbf{W}_3\,\mathrm{LayerNorm}(\mathbf{U})_{j,*}\big), && \text{for } j = 1 \ldots S.
\end{aligned}
$$

    where the first equation is the *token (spatial) mixing* operation and the second equation is the *channel mixing* operation.
    σ is the element-wise non-linearity GELU.
    D_S and D_C are the hidden-widths in the token-mixing and channel-mixing MLPs.
  - 🟡 **P3** Mixer uses a standard classification head with the global average pooling layer followed by a linear classifier.

  - **Comparison with other architectures (CNNs and ViTs)**:
    - *Mixing operations in CNNs*: 🟡 **P2** Modern deep vision architectures consist of layers that mix features (i) at a given spatial location,(ii) between different spatial locations, or both at once. In CNNs, (ii) is implemented with N × N convolutions (for N > 1) and pooling. Neurons in deeper layers have a larger receptive field [1 , 28]. At the same time, 1×1 convolutions also perform (i), and larger kernels perform both (i) and (ii).
    - *Mixing operations in ViTs*: 🟡 **P2** In Vision Transformers and other attention-based architectures, self-attention layers allow both (i) and (ii) and the MLP-blocks perform (i).
    - *Quadratic Self-attention cost w.r.t. input length vs Linear Cost*: 🟡 **P3** Note that DS is selected independently of the number of input patches. Therefore, the computational complexity of the network is linear in the number of input patches, unlike ViT whose complexity is quadratic.
      To see why this is the case: The transformation in the token-mixing MLP can be thought as $\text{Input} \in \mathbb{R}^{C \times S} \xrightarrow{W_1 \in \mathbb{R}^{S \times D_S}} \mathbb{R}^{C \times D_S} \xrightarrow{W_2 \in \mathbb{R}^{D_S \times S}} \text{Ouput} \in \mathbb{R}^{C \times S}$ where each operation is a

matrix-multiplication. Therefore, D_S is independent of the number of patches S.
🟡 **P3** Since DC is independent of the patch size, the overall complexity is linear in the number of pixels in the image, as for a typical CNN.

- *Positional Invariance*: 🟡 **P3** Tying the parameters of the channel-mixing MLPs (within each layer) is a natural choice—it provides positional invariance, a prominent feature of convolutions
- *Relation to Depthwise-Separable convolutions*: 🟡 **P3** However, tying parameters across channels is much less common. (Sharing weights across columns in token-mixing MLPs)
🟡 **P3** For example, separable convolutions [9, 40], used in some CNNs, apply convolutions to each channel independently of the other channels. However, in separable convolutions, a different convolutional kernel is applied to each channel unlike the token-mixing MLPs in Mixer that share the same kernel (of full receptive field) for all of the channels.
- 🟡 **P2** In the extreme case, our architecture can be seen as a very special CNN, which uses 1×1 convolutions for channel mixing, and single-channel depth-wise convolutions of a full receptive field and parameter sharing for token mixing.
- *Sparse Convolutions (Grouped or Depth-wise)*: 🟡 **P9** Mixer takes the idea of using convolutions with small kernels to the extreme: by reducing the kernel size to1×1 it turns convolutions into standard dense matrix multiplications applied independently to each spatial location (channel-mixing MLPs). This alone does not allow aggregation of spatial information and to compensate we apply dense matrix multiplications that are applied to every feature across all spatial locations (token-mixing MLPs). In Mixer, matrix multiplications are applied row-wise or column-wise on the "patches×features" input table, which is also closely related to the work on sparse convolutions.
- *Isotropic vs Pyramidal design*: 🟡 **P3** Each layer in Mixer (except for the initial patch projection layer) takes an input of the same size. This"isotropic" design is most similar to Transformers, or deep RNNs in other domains, that also use afixed width. This is unlike most CNNs, which have a pyramidal structure: deeper layers have a lower resolution input, but more channels.
- *Positional Embeddings*: 🟡 **P3** Unlike ViTs, Mixer does not use position embeddings because the token-mixing MLPs are sensitive to the order of the input tokens.
- Both ViTs and Mixer are restricted by the input resolution i.e., the architecture is dependent upon input resolution whereas for CNN this is not the case.

- **Fine-tuning**:
  - Fine-tuning is done at a higher resolution than the pre-training resolution. The image patch size is kept constant, therefore, there is an increase in the number of tokens. How to make the token-mixing MLPs handle these longer sequences?
  - 🟡 **P14** For simplicity we assume that the image resolution is increased by an integer factor K. The length S of the token sequence increases by a factor of K2. We increase the hidden width DS of the token-mixing MLP by a factor of K2 as well. Now we need to initialize the parameters of this new(larger) MLP with the parameters of the pre-trained MLP. To this end we split the input sequence into K2 equal parts, each one of the original length S, and initialize the new MLP so that it processes all these parts independently in parallel with the pre-trained MLP. Formally, the pre-trained weight matrix W1 ∊ RDS ×S of the original MLP in Eq. 1 of the main text will be now replaced with a larger matrix W'1 ∊ R(K2·DS)×(K2·S). Assume the token sequence for the resized input image is a concatenation of K2 token sequences of length S each, computed by splitting the input into K × K equal parts spatially. We then initialize W'1 with a block-diagonal matrix that has copies of W1 on its main diagonal. Other parameters of the MLP are handled analogously.

- **Results**:
  - Architectures are compared on the following basis:
    - Accuracy on the downstream task
    - Total computational cost of pre-training, which is important when training the model from scratch on the upstream dataset
    - Test-time throughput, which is important for the practitioner
  - **Notation**: The naming convention for Mixer architecture follows ViTs. 🟡 **P4** A brief notation "B/16" means the model of base scale with patches of resolution 16×16.
-
  - 🟡 **P5** Overall, Figure 2 (left) supports our main claim that in terms of the accuracy-compute trade-off Mixer is competitive with more conventional neural network architectures.
  - *Training Dataset Size*: 🟡 **P7** The same conclusions hold for ViT, consistent with Dosovitskiy et al. [14] . However, the relative improvement of larger Mixer models are even more pronounced. The performance gap between Mixer-L/16 and ViT-L/16 shrinks with data scale. It appears that Mixer benefits from the growing dataset size even more than ViT. One could speculate and explain it again with the difference in inductive biases: self-attention layers in ViT lead to certain properties of the learned functions that are less compatible with the true underlying distribution than those discovered with Mixer architecture.
-

- *Permutation Invariance*:
  - Train Mixer-B/16 and ResNet50×1 model with two different input transformations:
    - 🟡 **P8** Shuffle the order of 16×16 patches and permute pixels within each patch with a shared permutation;
    - 🟡 **P8** Permute the pixels globally in the entire image.

    🟡 **P8** Same permutation is used across all images.
  - Conclusion:
    - 🟡 **P8** Mixer is invariant to the order of patches and pixels within the patches (the blue and green curves match perfectly). On the other hand, ResNet's strong inductive bias relies on a particular order of pixels within an image and its performance drops significantly when the patches are permuted.
    - 🟡 **P8** Remarkably, when globally permuting the pixels, Mixer's performance drops much less (~45% drop) compared to the ResNet (~75% drop).
  -

	Permuting pixels locally	**Permute patches**	**Globally permute pixels**
CNNs	✓	✗	✗
ViTs	✓	✓	✗
Mixer	✓	✗	✗

Note:
- The ✓ represents whether the architecture is invariant to that particular input transformation or not.
- CNNs will be invariant to local pixel permutation only when the permutation happens within a patch whose size is less than the least kernel size in the CNN.

- *Visual Inspection*:
  - Inspection of weight matrices $W_1 \in \mathbb{R}^{D_S \times S}$.
    - Note that each column of W_1 is treated as a separate hidden unit which is being visualized. For Mixer-B/16, W_1 has the shape `(384, 196)` because there are 14×14 patches.

      🟡 **P8**

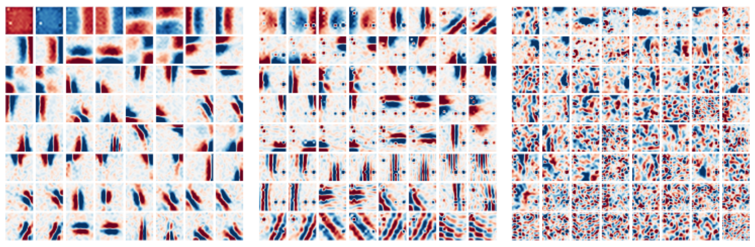

Figure 5: Hidden units in the first (**left**), second (**center**), and third (**right**) token-mixing MLPs of a Mixer-B/16 model trained on JFT-300M. Each unit has 196 weights, one for each of the 14×14 incoming patches. We pair the units to highlight the emergence of kernels of opposing phase. Pairs are sorted by filter frequency. In contrast to the kernels of convolutional filters, where each weight corresponds to one pixel in the input image, one weight in any plot from the left column corresponds to a particular 16×16 patch of the input image. Complete plots in Supplementary D.

      Note here only 64 of the 384 hidden units have been shown for each of the MLP. For full plots refer to the Supplementary D. Also, 🟡 **P14** For better visualization, we sort all hidden units according to a heuristic that tries to show low frequency filters first. For each unit, we also try to identify the unit that is closest to its inverse.
    - *Conclusions*:
      - 🟡 **P8** Some of the learned features operate on the entire image, while others operate on smaller regions. Deeper layers appear to have no clearly identifiable structure. Therefore, as expected Mixer allows global exchange of information from the first layer itself.
      - 🟡 **P8** Similar to CNNs, we observe many pairs of feature detectors with opposite phases [39].
  - Inspection of the projection matrices (which projects the input patch).
    -

🟡 **P15**

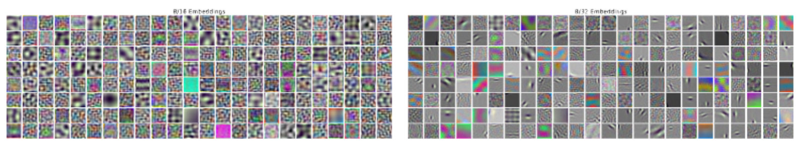

Figure 7: Linear projection units of the embedding layer for Mixer-B/16 (**left**) and Mixer-B/32 (**right**) models pre-trained on JFT-300M. Mixer-B/32 model that uses patches of higher resolution 32 × 32 learns very structured low frequency projection units, while most of the units learned by the Mixer-B/16 have high frequencies and no clear structure.

Here the projection matrix $P \in \mathbb{R}^{(P_S \cdot P_S \cdot 3) \times C}$ where P_S is the patch size. So, the authors are visualizing each column of the matrix.

- 🟡 **P14** nterestingly, it appears that their properties strongly depend on the patch resolution used by the models. Across all Mixer model scales, using patches of higher resolution 32×32 leads to Gabor-like low-frequency linear projection units, while for the 16×16 resolution the units show no such structure.

- *Modifying the token-mixing MLPs*:
  - *Untying (not-sharing) the parameters (Making the token-mixing more similar to depthwise-separable convolution)*: 🟡 **P12** Token-mixing MLPs in the Mixer layer are shared across the columns of the input table X ∊ RS×C . In other words, the same MLP is applied to each of the C different features. Instead, we could introduce C separate MLPs with independent weights, effectively multiplying the number of parameters by C. We did not observe any noticeable improvements.
  - *Grouping Channels Together*:
    - *Making Mixers more similar to Group Convolutions*: 🟡 **P13** Token-mixing MLPs take S-dimensional vectors as inputs. Every such vector contains values of a single feature across S different spatial locations. In other words, token-mixing MLPs operate by looking at only one channel at once. One could instead group channels together by concatenating G neighbouring columns in X ∊ RS×C , reshaping it to a matrix of dimension (S · G) × (C/G). This increases the MLP's input dimensionality from S to G · S and reduces the number of vectors to be processed from C to C/G. Now the MLPs look at several channels at once when mixing the tokens. This concatenation of the column-vectors improved linear5-shot top-1 accuracy on ImageNet by less than 1–2%.
    - *Making Mixers more similar to Multi-head Attention*: 🟡 **P13** We tried a different version, where we replace the simple reshaping described above with the following: (1) Introduce G linear functions (with trainable parameters) projecting RC to RC/G. (2) Using them, map each of the S rows (tokens) in X ∊ RS×C to G different (C/G)-dimensional vectors. This results in G different "views" on every token, each one consisting of C/G features.(3) Finally, concatenate vectors corresponding to G different views for each of the C/G features. This results in a matrix of dimension (S · G) × (C/G). The idea is that MLPs can look at G different views of the original channels, when mixing the tokens. This version improved the top-5 ImageNet accuracy by 3–4% for the Mixer-S/32 architecture, however did not show any improvements for the larger scales.
  - *Pyramidal Structure for Mixers*: 🟡 **P13** We tried using the token-mixing MLP to reduce the number of tokens by mapping from S input tokens to S′ < S output tokens. While first experiments showed that on JFT-300M such models significantly reduced training time without losing much performance, we were unable to transfer these findings to ImageNet or ImageNet-21k.