RepMLPNet: Hierarchical Vision MLP with Re-parameterized Locality

Xiaohan Ding [1] * Honghao Chen [2] Xiangyu Zhang [3] Jungong Han [4] Guiguang Ding [1†]
[1] Beijing National Research Center for Information Science and Technology (BNRist);
School of Software, Tsinghua University, Beijing, China
[2] Institute of Automation, Chinese Academy of Sciences
[3] MEGVII Technology
[4] Computer Science Department, Aberystwyth University, SY23 3FL, UK
dxh17@mails.tsinghua.edu.cn chenhonghao2021@ia.ac.cn zhangxiangyu@megvii.com
jungonghan77@gmail.com dinggg@tsinghua.edu.cn

Abstract

Compared to convolutional layers, fully-connected (FC) layers are better at modeling the long-range dependencies but worse at capturing the local patterns, hence usually less favored for image recognition. In this paper, we propose a methodology, Locality Injection, to incorporate local priors into an FC layer via merging the trained parameters of a parallel conv kernel into the FC kernel. Locality Injection can be viewed as a novel Structural Re-parameterization method since it equivalently converts the structures via transforming the parameters. Based on that, we propose a multi-layer-perceptron (MLP) block named RepMLP Block, which uses three FC layers to extract features, and a novel architecture named RepMLPNet. The hierarchical design distinguishes RepMLPNet from the other concurrently proposed vision MLPs. As it produces feature maps of different levels, it qualifies as a backbone model for downstream tasks like semantic segmentation. Our results reveal that 1) Locality Injection is a general methodology for MLP models; 2) RepMLPNet has favorable accuracy-efficiency trade-off compared to the other MLPs; 3) RepMLPNet is the first MLP that seamlessly transfer to Cityscapes semantic segmentation. The code and models are available at https://github.com/DingXiaoH/RepMLP.

1. Introduction

The locality of images (*i.e.*, a pixel is more related to its neighbors than the distant pixels) makes Convolutional Neural Network (ConvNet) successful in image recognition, as a conv layer only processes a local neighborhood. In this paper, we refer to this inductive bias as the *local prior*.

On top of that, we also desire the ability to capture the long-range dependencies, which is referred to as the *global capacity* in this paper. Traditional ConvNets model the long-range dependencies by the large receptive fields formed by deep stacks of conv layers [36]. However, repeating local operations is computationally inefficient and may cause optimization difficulties. Some prior works enhance the global capacity with self-attention-based modules [15, 35, 36], which has no local prior. For example, ViT [15] is a pure-Transformer model without convolution, which feeds images into the Transformers as a sequence. Due to the lack of local prior as an important inductive bias, ViT needs an enormous amount of training data (3×10^8 images in JFT-300M) to converge. On the other hand, a fully-connected (FC) layer can also directly model the dependencies between any two input points if we flatten the feature map as a vector, linearly map it into another vector, and reshape the resultant vector back into a feature map. However, this process has no locality either. Without such an inductive bias, the concurrently proposed MLPs [22, 32, 33, 39] usually demand a huge amount of training data (*e.g.*, JFT-300M), an extended training epochs (300 or 400 ImageNet [7] epochs) or special training techniques (*e.g.*, a DeiT-style distillation method [33, 34]) to *learn the inductive bias from scratch*. Otherwise, it cannot reach a comparable level of performance with traditional ConvNets.

Another drawback of the concurrently proposed MLPs is the difficulty of transferring to the downstream tasks like semantic segmentation. For example, MLP-Mixer [32] demonstrates satisfactory performance on ImageNet but does not qualify as the backbone of a segmentation framework like UperNet [37] as it aggressively embeds (*i.e.*,

*This work is supported by The National Key Research and Development Program of China (No. 2017YFA0700800), the National Natural Science Foundation of China (No.61925107, No.U1936202) and Beijing Academy of Artificial Intelligence (BAAI). This work is done during Xiaohan Ding and Honghao Chen's internship at MEGVII.

†Corresponding author.

downsamples) the images by 16× and repeatedly transforms the embeddings, so that it cannot produce multi-scale feature maps with different levels of semantic information.

This paper aims at an MLP model that is **1)** friendly to small-data, **2)** trainable with ordinary training methods, and **3)** transferrable to downstream tasks like semantic segmentation. To his end, we make contributions on three levels: methodology, component and architecture.

Methodology. We propose a novel methodology, Locality Injection, to provide an FC layer with what it demands for effective visual understanding: the locality. Specifically, we place one or more conv layers parallel to the FC and add up their outputs. Though the FC simply views the feature maps as vectors, completely ignoring the locality, the conv layers can capture the local patterns. However, though such conv layers bring only negligible parameters, the inference speed may be considerably slowed down because of the extra reshaping operations and the reduction of degree of parallelism on high-power computing devices like GPU [26]. Therefore, we propose to equivalently *merge the conv layers into the FC kernels* after training to speed up the inference. By doing so, we obtain an FC layer that is structurally identical to a normally trained FC layer (*i.e.*, identical inference cost) but is parameterized by a special matrix with locality. Since Locality Injection converts the training-time structure (*i.e.*, FC + conv) to the inference-time (*i.e.*, a single FC) via an equivalent transformation on the parameters, it can be viewed as a novel Structural Re-parameterization [14] technique. In other words, we *equivalently incorporate the inductive bias into a trained FC layer, instead of letting it learn from scratch*. The key to such equivalent transformation is converting an arbitrary conv kernel to an FC kernel (*i.e.*, a Toeplitz matrix). In this paper, we propose a simple, platform-agnostic and differentiable approach, which will be introduced in Sect. 3.

Component. Based on Locality Injection, we propose RepMLP Block as an MLP building block. Fig. 1 shows a training-time RepMLP Block with FC, conv and batch normalization [18] (BN) layers can be equivalently converted into an inference-time block with only three FC layers.

Architecture. With RepMLP Block, we propose a novel MLP architecture with a hierarchical design, which produces semantic information on different levels, so that it can be readily used as the backbone of the common downstream frameworks. In other words, as the feature map size reduces, the number of channels expands. We reckon that the major obstacle for adopting hierarchical design in a ResMLP- or MLP-Mixer-style MLP model is that the number of parameters is coupled with the feature map size [1], so that the number of parameters of the lower-level FC layers would be several orders of magnitude greater than the higher-level layers. For example, assume the lowest-level feature maps are of 56×56, an FC layer requires $56^4 = 9.8$M parameters for mapping a channel to another, without any cross-channel correlations (*i.e.*, like depthwise convolution). We may let all the channels share the same set of parameters, so that the layer will have a total of 9.8M parameters. However, let the highest-level feature maps be of 7×7, the parameter count is only $7^4 = 2.4$K, and the number of channels is large. Predictably, sharing so few parameters among so many channels restricts the representational capacity hence results in inferior performance. We solve this problem by a set-sharing linear mapping (introduced in Sect. 4.1), so that we can independently control the parameter count of each layer by letting the channels share a configurable number of parameter sets s. With a smaller s for the lower-level layers and a larger s for the higher-level ones, we can balance the model size and the representational capacity.

In summary, with such a general methodology, a powerful building blck and a novel architecture, RepMLPNet achieves favorable accuracy-efficiency trade-off with only 100 training epochs on ImageNet, compared to the other MLP models trained in 300 or 400 epochs, and shows satisfactory performance as the first attempt to transfer an MLP-style backbone to semantic segmentation.

2. Related Work

2.1. From Vision Transformer to MLP

Vision Transformers heavily adopt self-attention layers to capture the spatial patterns. Compared to ConvNets, the recent transformer models [15, 23, 34] have achieved comparable or better performance. A primary motivation for using Transformers on vision tasks is to reduce the inductive bias designed by human and let the model automatically learn a better induct bias from large-scale data, which has been proved successful. Concurrently, MLP-Mixer [32], ResMLP [33] and gMLP [22] present a simpler MLP architecture to reduce the inductive biases even further. For example, MLP-Mixer alternatively mix the information across channels (chanenl-mixing, implemented by 1×1 conv) and within channels (token-mixing). Specifically, the token-mixing component projects the feature maps along the spatial dimension (*i.e.*, transpose the feature map tensor), feed them into a 1×1 conv, and transpose the outcomes back. Therefore, the token-mixing can be viewed as an FC layer that flattens every channel as a vector, completely ignoring the spatial information, linearly maps it into another vector, and reshapes it back into a channel; and all the channels share the same kernel matrix. In this way, MLP-Mixer realizes the communications between spatial locations. Ex-

[1] In this paper, MLP refers to a model that mostly uses FC layers to linearly map features from a vector to another, so that the number of parameters must be proportional to the input size and output size. By our definition, another model, CycleMLP [2], is not referred to as an MLP.

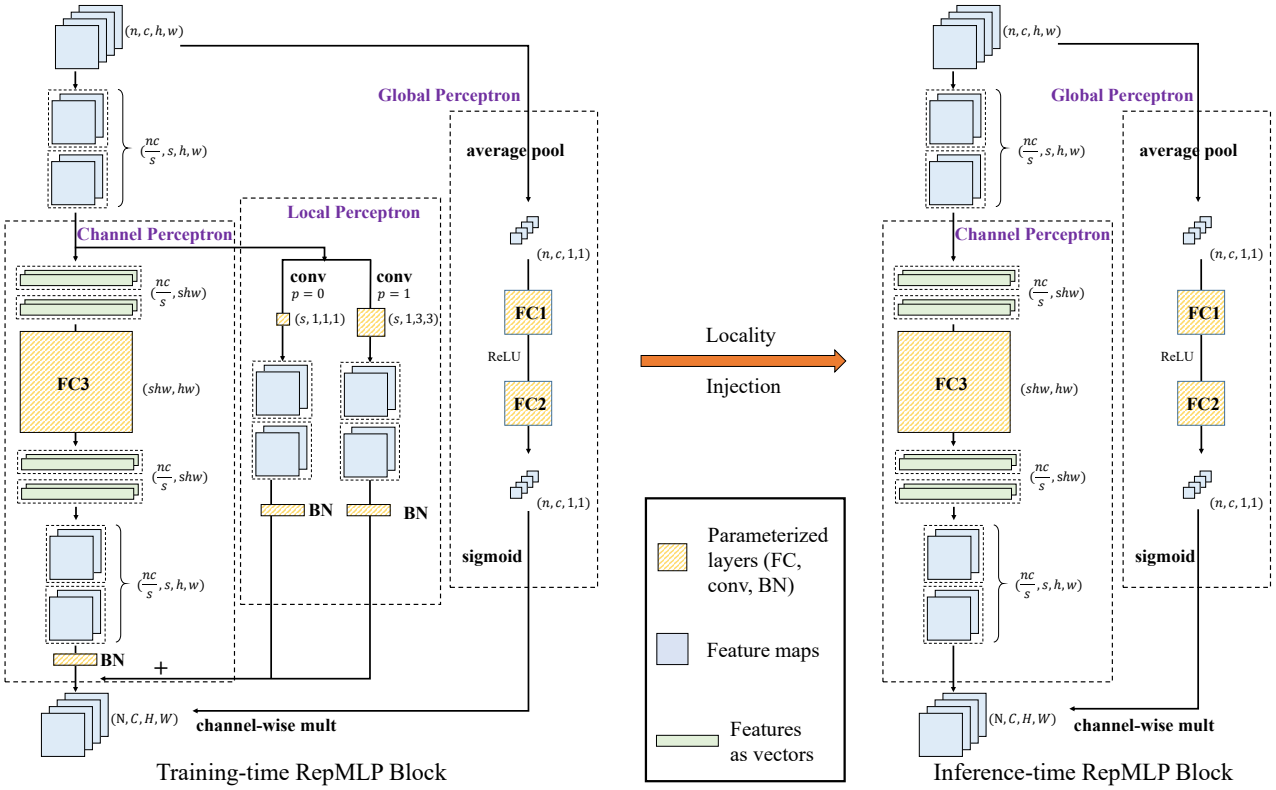


Figure 1. RepMLP Block, where n, c, h, w are the batch size, number of input channels, height and width of the feature map, s is the number of *share-sets*, p is the padding. This example assumes $n = 1$, $c = 4$, $s = 2$. **1)** The Global Perceptron aggregates the information across all the spatial locations and establishes the relations among channels. **2)** In parallel, the input feature map is split into two share-sets and fed into the Channel Perceptron, which simply reshapes the features into vectors, linearly maps it to the output vectors, and reshapes them back. **3)** The Local Perceptron takes the same inputs as the Channel Perceptron but convolve with small kernels to capture the local patterns. This example uses 1×1 and 3×3 so that the padding should be $p = 0$ and 1, respectively, to maintain the feature map size. Through batch normalization (BN) [18], the outputs of Local Perceptron and Channel Perceptron and added up. Finally, we combine the global and channel-wise information by merging the outcomes of the Global Perceptron. After training, the conv layers are absorbed into the FC3 kernel via Locality Injection, so that the training-time block is equivalently converted into a three-FC block and used for inference.

cept for MLP-Mixer, ResMLP [33] and gMLP [22] present different architectures which mixes the spatial information with a similar mechanism.

2.2. Structural Re-parameterization

The core of Locality Injection is to equivalently merge a trained conv kernel into a trained FC kernel to inject the locality. In this sense, it can be categorized into Structural Re-parameterization, which is a family of methodologies that convert structures via transforming the parameters. A representative prior work of Structural Rre-parameterization is RepVGG [14], which is a VGG-like architecture, as its body uses only 3×3 conv and ReLU for inference. Such an inference-time architecture is equivalently converted from a training-time architecture with identity and 1×1 branches. Asymmetric Convolution Block (ACB) [10] and Diverse Branch Block (DBB) [13] are two replacements for regular conv layers. Via constructing extra training-time paths (*e.g.*, 1×3, 3×1, or 1×1-3×3), they can improve a regular ConvNet without extra inference costs. ResRep [12] uses Structural Re-parameterization for channel pruning [8, 9, 11, 21, 24] and achieves state-of-the-art results, which reduces the filters in a conv layer via constructing and pruning a following 1×1 layer.

Locality Injection can be viewed as a remarkable attempt to generalize the idea of Re-parameterization beyond convolution. By merging a conv into a FC kernel, we bridge conv and FC with a simple, platform-agnostic and differentiable method (Sect. 3).

3. Locality Injection via Re-parameterization

3.1. Formulation

In this paper, a feature map is denoted by a tensor $\mathrm{M} \in \mathbb{R}^{n \times c \times h \times w}$, where n is the batch size, c is the number of channels, h and w are the height and width, respectively.

We use F and W for the kernel of conv and FC, respectively. For the simplicity and ease of re-implementation, we use the same data format as PyTorch [29] and formulate the transformations in a pseudo-code style. For example, the data flow through a $k \times k$ conv is formulated as

$$\mathrm{M}^{(\mathrm{out})} = \mathrm{CONV}(\mathrm{M}^{(\mathrm{in})}, \mathrm{F}, p)\,, \tag{1}$$

where $\mathrm{M}^{(\mathrm{out})} \in \mathbb{R}^{n \times o \times h' \times w'}$ is the output feature map, o is the number of output channels, p is the number of pixels to pad, $\mathrm{F} \in \mathbb{R}^{o \times c \times k \times k}$ is the conv kernel (we temporarily assume the conv is dense, *i.e.*, the number of groups is 1). From now on, we assume $h' = h, w' = w$ for the simplicity (*i.e.*, the stride is 1 and $p = \lfloor \frac{k}{2} \rfloor$).

For an FC, let p and q be the input and output dimensions, $\mathrm{V}^{(\mathrm{in})} \in \mathbb{R}^{n \times p}$ and $\mathrm{V}^{(\mathrm{out})} \in \mathbb{R}^{n \times q}$ be the input and output, respectively, the kernel is $\mathrm{W} \in \mathbb{R}^{q \times p}$ and the matrix multiplication (MMUL) is formulated as

$$\mathrm{V}^{(\mathrm{out})} = \mathrm{MMUL}(\mathrm{V}^{(\mathrm{in})}, \mathrm{W}) = \mathrm{V}^{(\mathrm{in})} \cdot \mathrm{W}^\intercal\,. \tag{2}$$

We now focus on an FC that takes $\mathrm{M}^{(\mathrm{in})}$ as input and outputs $\mathrm{M}^{(\mathrm{out})}$ and assume the output shape is the same as the input. We use RS (short for "reshape") as the function that only changes the shape specification of tensors but not the order of data in memory, which is *cost-free*. The input is first flattened into n vectors of length chw, which is $\mathrm{V}^{(\mathrm{in})} = \mathrm{RS}(\mathrm{M}^{(\mathrm{in})}, (n, chw))$, multiplied by the kernel $\mathrm{W}(ohw, chw)$, then the output $\mathrm{V}^{(\mathrm{out})}(n, ohw)$ is reshaped back into $\mathrm{M}^{(\mathrm{out})}(n, o, h, w)$. For the better readability, we omit the RS if there is no ambiguity,

$$\mathrm{M}^{(\mathrm{out})} = \mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W})\,. \tag{3}$$

Such an FC cannot take advantage of the locality of images as it computes each output point according to every input point, unaware of the positional information.

3.2. Locality Injection

Assume there is a conv layer parallel to the FC (like the Channel Perceptron and Local Perceptron shown in Fig. 1), which takes $\mathrm{M}^{(\mathrm{in})}$ as input and outputs $\mathrm{M}^{(\mathrm{out})}$, we describe how to equivalently merge it into the FC. In the following, we assume the FC kernel is $\mathrm{W}^{(1)}(ohw, chw)$, conv kernel is $\mathrm{F}(o, c, k, k)$ and padding is p. Formally, we desire to construct W' so that

$$\begin{aligned}&\mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}') \\ &= \mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(1)}) + \mathrm{CONV}(\mathrm{M}^{(\mathrm{in})}, \mathrm{F}, p)\,.\end{aligned} \tag{4}$$

We note that for any kernel $\mathrm{W}^{(2)}$ of the same shape as $\mathrm{W}^{(1)}$, the additivity of MMUL ensures that

$$\begin{aligned}&\mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(1)}) + \mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(2)}) \\ &= \mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(1)} + \mathrm{W}^{(2)})\,.\end{aligned} \tag{5}$$

Therefore, we can merge F into $\mathrm{W}^{(1)}$ if we can construct $\mathrm{W}^{(\mathrm{F},p)}$ of the same shape as $\mathrm{W}^{(1)}$ which satisfies

$$\mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(\mathrm{F},p)}) = \mathrm{CONV}(\mathrm{M}^{(\mathrm{in})}, \mathrm{F}, p)\,. \tag{6}$$

Obviously, $\mathrm{W}^{(\mathrm{F},p)}$ must exist, since a conv can be viewed as a sparse FC that shares parameters among spatial positions (*i.e.*, a Toeplitz matrix), which is exactly the source of its translation invariance, but it is not obvious to construct it with given F and p. As modern computing platforms use different algorithms of convolution (*e.g.*, im2col- [1], Winograd- [20], FFT- [27], MEC- [3], and sliding-window-based) and the memory allocation of data and implementations of padding may be different, a means for constructing the Toeplitz matrix on a specific platform may not work on another platform. In this paper, we propose a simple and *platform-agnostic* solution.

As discussed above, for *any* input $\mathrm{M}^{(\mathrm{in})}$ and conv kernel F, padding p, there exists an FC kernel $\mathrm{W}^{(\mathrm{F},p)}$ such that

$$\mathrm{M}^{(\mathrm{out})} = \mathrm{CONV}(\mathrm{M}^{(\mathrm{in})}, \mathrm{F}, p) = \mathrm{MMUL}(\mathrm{M}^{(\mathrm{in})}, \mathrm{W}^{(\mathrm{F},p)})\,. \tag{7}$$

With the formulation used before (Eq. 2), we have

$$\mathrm{V}^{(\mathrm{out})} = \mathrm{V}^{(\mathrm{in})} \cdot \mathrm{W}^{(\mathrm{F},p)\intercal}\,. \tag{8}$$

We insert an identity matrix $\mathrm{I}\,(chw, chw)$ and use the associative law

$$\mathrm{V}^{(\mathrm{out})} = \mathrm{V}^{(\mathrm{in})} \cdot (\mathrm{I} \cdot \mathrm{W}^{(\mathrm{F},p)\intercal})\,. \tag{9}$$

With explicit RS, we rewrite Eq. 9 as

$$\mathrm{V}^{(\mathrm{out})} = \mathrm{V}^{(\mathrm{in})} \cdot \mathrm{RS}(\mathrm{I} \cdot \mathrm{W}^{(\mathrm{F},p)\intercal}, (chw, ohw))\,. \tag{10}$$

We note that $\mathrm{W}^{(\mathrm{F},p)}$ is constructed with an existing conv kernel F, so that $\mathrm{I} \cdot \mathrm{W}^{(\mathrm{F},p)\intercal}$ *is exactly a convolution* with F on a feature map $\mathrm{M}^{(\mathrm{I})}$ which is reshaped from I. That is

妙极！ $$\mathrm{I} \cdot \mathrm{W}^{(\mathrm{F},p)\intercal} = \mathrm{CONV}(\mathrm{M}^{(\mathrm{I})}, \mathrm{F}, p), \tag{11}$$

where

$$\mathrm{M}^{(\mathrm{I})} = \mathrm{RS}(\mathrm{I}, (chw, c, h, w))\,. \tag{12}$$

Comparing Eq. 8 with Eq. 11, 10, we have

$$\mathrm{W}^{(\mathrm{F},p)} = \mathrm{RS}(\mathrm{CONV}(\mathrm{M}^{(\mathrm{I})}, \mathrm{F}, p), (chw, ohw))^\intercal, \tag{13}$$

which is exactly the expression we desire for constructing $\mathrm{W}^{(\mathrm{F},p)}$ with F, p. In short, the equivalent FC kernel of a conv kernel is the result of convolution on an identity matrix with proper reshaping. Better still, the conversion is efficient and *differentiable*, so one may derive the FC kernel during training and use it in the objective function (*e.g.*, for penalty-based pruning [16, 28]). The expression and code for the groupwise case are derived in a similar way and provided in the supplementary material.

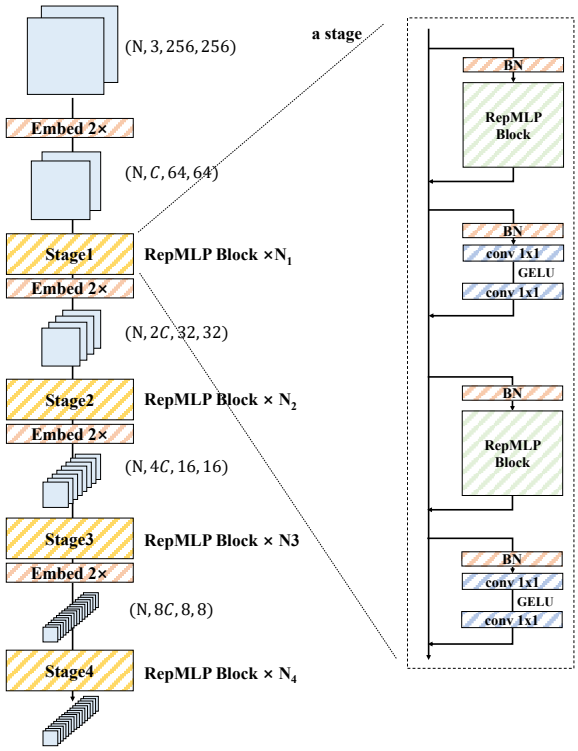


Figure 2. Architecture of RepMLPNet.

4. RepMLPNet

RepMLPNet is a hierarchical MLP-style architecture composed RepMLP Blocks. We first introduce RepMLP Block (Fig. 1) in Sect. 4.1 and then describe the overall architecture (Fig. 2) in Sect. 4.2.

4.1. Components of RepMLP Block

A training-time RepMLP Block is composed of three parts termed as Global Perceptron, Channel Perceptron and Local Perceptron (Fig. 1), which are designed to model the information on different levels. Specifically, Global Perceptron aims at modeling the coarse global dependencies across spatial localtions among all the channels; Channel Perceptron is designed for modeling the long-range spatial dependencies within each channel; Local Perceptron is responsible for capturing the local patterns. The outputs of the three components are combined to obtain a comprehensive transformation of the input features.

Global Perceptron average-pools the inputs (n, c, h, w) into vectors $(n, c, 1, 1)$ and feed them into two FC layers to obtain a vector that encodes the global information.

Channel Perceptron contains an FC layer that directly performs the feature transformation, where the key is the *set-sharing* mechanism. We follow the formulation used in Sect. 3 and assume $o = c$ for the convenience. We note that a normal FC layer has $(chw)^2$ parameters. Taking a 64×64 feature map with 128 channels for example, the parameter count of a normal dense FC will be 2.1B, which is clearly unacceptable. A natural solution is to make the FC "depthwise" just like depthwise conv, which will not be able to model cross-channel dependencies but has only $1/c$ parameters and FLOPs. However, even a parameter count of $c(hw)^2$ is too large. Our solution is to make multiple channels share a set of spatial-mapping parameters, so that the parameters are reduced to $s \times (hw)^2$, where s is the number of *share-sets* of parameters. In other words, every $\frac{c}{s}$ channels share the same set of parameters, and there are s such share-sets in total. Specifically, we first evenly split the c channels into c/s groups, which means $(n, c, h, w) \rightarrow (\frac{nc}{s}, s, h, w)$, and then flatten it into $\frac{nc}{s}$ vectors, which each have a length of shw, feed the vectors into a "depthwise" FC, and reshape the output back. Compared to "depthwise" FC, set-sharing FC not only breaks the correlation between channels ($(chw)^2$ parameters $\rightarrow c(hw)^2$), but reduces the parameters even further ($c(hw)^2$) $\rightarrow s(hw)^2$); but it does not reduce the computations compared to a "depthwise" FC. It should be noted that the spatial mappings in ResMLP and MLP-Mixer are implemented in a different way (transpose, 1×1 conv and transpose back) but are mathematically equivalent to a special case of set-sharing FC with $s = 1$, which means all the channels share the same $(hw)^2$ parameters. We will show that increasing s can improve the performance with more parameters but no extra FLOPs, which is useful in scenarios where the model size is not a major concern (Table. 3).

In practice, though set-sharing FC is not directly supported by some computing frameworks like PyTorch, it can be alternatively implemented by a groupwise 1×1 conv. Formally, let $\mathrm{V}^{(\mathrm{in})}(\frac{nc}{s}, shw)$ be the vectors split in share-sets, the implementation is composed of three steps: **1)** reshaping $\mathrm{V}^{(\mathrm{in})}$ as a "feature map" with spatial size of 1×1, which is $(\frac{nc}{s}, shw, 1, 1)$; **2)** performing 1×1 conv with s groups (so that the parameters are $(shw)^2/s = s(hw)^2$); **3)** reshaping the output into $(\frac{nc}{s}, s, h, w)$, then (n, c, h, w).

Local Perceptron comprises conv branches that take the same outputs as Channel Perceptron and BN layers follow the conv layers as a common practice. Each conv is depthwise and has the same number of share-sets s on the s-channel feature maps, so the parameter shape is $(s, 1, K, K)$.

Merging the Local Perceptron into Channel Perceptron via Locality Injection requires the elimination of the BN layers, so that we first equivalently fusing them into the preceding conv layers and FC3. Please note that the conv layers are depthwise and the number of input channels is s. Following the PyTorch format again, let $\mathrm{F} \in \mathbb{R}^{s \times 1 \times k \times k}$ be the conv kernel, $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^s$ be the accumulated mean, standard deviation and learned scaling factor and bias of the

following BN, we construct the kernel F' and bias $\mathbf{b}'$ as

$$\mathrm{F}'_{i,:,:,:} = \frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i}\mathrm{F}_{i,:,:,:}\,, \quad \mathbf{b}'_i = -\frac{\boldsymbol{\mu}_i\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i} + \boldsymbol{\beta}_i\,. \tag{14}$$

Then it is easy to verify the equivalence:

$$\begin{aligned}&\frac{\boldsymbol{\gamma}_i}{\boldsymbol{\sigma}_i}(\mathrm{CONV}(\mathrm{M},\mathrm{F},p)_{:,i,:,:} - \boldsymbol{\mu}_i) + \boldsymbol{\beta}_i \\ &= \mathrm{CONV}(\mathrm{M},\mathrm{F}',p)_{:,i,:,:} + \mathbf{b}'_i\,, \forall 1 \leq i \leq s\,,\end{aligned} \tag{15}$$

where the left side is the original computation flow of a conv-BN, and the right is the constructed conv with bias.

The FC3 and BN in Channel Perceptron are fused in a similar way into $\hat{\mathrm{W}} \in \mathbb{R}^{shw \times hw}$, $\hat{\mathbf{b}} \in \mathbb{R}^{shw}$. Then we convert every conv via Eq. 13 and add the resultant matrix onto $\hat{\mathrm{W}}$. The biases of conv are simply replicated by hw times (because all the points on the same channel share a bias value) and added onto $\hat{\mathbf{b}}$. Finally, we obtain a single FC kernel and a single bias vector, which will be used to parameterize the inference-time FC3.

4.2. Hierarchical Architectural Design

Some recent vision MLP models have a similar design pattern: downsampling the inputs aggressively (*e.g.*, by 16×16) at the very beginning, and stacking multiple blocks to process the resultant small-sized features. In contrast, we adopt a hierarchical design paradigm, which has proved effective by previous studies on ConvNets [17, 30, 31, 38] and Transformers [23] but has not been used on vision MLPs, to the best of our knowledge.

Specifically, we arrange multiple RepMLP Blocks in four stages. The blocks within the same stage share the same set of structural hyper-parameters. At first, the input images are downsampled by $4\times$ with an embedding layer, which is implemented by a conv layer with a kernel size of 4×4 and stride of 4. From a stage to the next, we use an embedding layer to halve the width and height of feature maps and double the channels. Therefore, a RepMLPNet can be instantiated with the following hyper-parameters: the number of RepMLP Blocks in each stage $[B_1, B_2, B_3, B_4]$, the number of channels of the first stage C (so that the four stages will have C, $2C$, $4C$, $8C$ channels, respectively), the input resolution $H \times W$ (so that $h_1 = H/4, w_1 = W/4, ..., h_4 = H/32, w_4 = W/32$), and the number of share-sets of each stage ($[S_1, S_2, S_3, S_4]$). Considering the number of parameters of FC3 is $s(hw)^2$, we use a smaller s at the early stages.

An advantage of our hierarchical architecture is that the feature maps produced by any stage can be readily used to downstream tasks. For example, UperNet [37] requires four levels of feature maps with different sizes, so that it cannot use MLP-Mixer or ResMLP as the backbone. In contrast, combining RepMLPNet and UperNet or other downstream frameworks is as easy as using traditional ConvNets.

Table 1. Architectural hyper-parameters of RepMLPNet models (T for tiny, B for base, D for deep, L for large). Of note is that RepMLP-D256 is deeper than RepMLP-B256 but narrower so that they have comparable FLOPs and number of parameters.

Name	Input resolution	**B**	C	**S**
RepMLP-T224	224×224	$[2, 2, 6, 2]$	64	$[1, 4, 16, 128]$
RepMLP-B224	224×224	$[2, 2, 12, 2]$	96	$[1, 4, 32, 128]$
RepMLP-T256	256×256	$[2, 2, 6, 2]$	64	$[1, 4, 16, 128]$
RepMLP-B256	256×256	$[2, 2, 12, 2]$	96	$[1, 4, 32, 128]$
RepMLP-D256	256×256	$[2, 2, 18, 2]$	80	$[1, 4, 16, 128]$
RepMLP-L256	256×256	$[2, 2, 18, 2]$	96	$[1, 4, 32, 256]$

5. Experiments

5.1. ImageNet Classification

We first instantiate a series of RepMLPNets with different architectural hyper-parameters, as shown in Table. 1.

We evaluate RepMLPNets on ImageNet [7] classification. All the RepMLPNets are trained with identical settings: a global batch size of 256 distributed on 8 GPUs, AdamW [25] optimizer with initial learning rate of 0.002, weight decay of 0.1 and momentum coefficient of 0.9. We train for only *100 epochs* in total with cosine learning rate annealing, including a 10-epoch warm-up at the beginning. We use label smoothing of 0.1, mixup [41] with $\alpha = 0.4$, CutMix [40] with $\alpha = 1.0$, and RandAugment [6]. As a series of strong baselines, we present several ConvNets trained with a *strong scheme* with RandAugment, mixup, label smoothing, and SGD optimizer with weight decay of 4×10^{-5}. We would like the comparison to be slightly biased towards the traditional ConvNets, so we train them for 120 epochs. All the models are evaluated with single central crop and the throughput is tested on the same 2080Ti GPU with a batch size of 128.

It should be noted that most of the results reported by [22, 32, 33, 39] are produced with a long training schedule of 300 or 400 epochs, or an advanced distillation method (the DeiT-style training [34]). Therefore, except for the results cited from the corresponding papers [22, 32, 33, 39], we train an MLP-Mixer and a ResMLP-S12 with simple training settings for a fair comparison (labeled as "our impl" in Table. 2). Specifically, the MLP-Mixer is trained with the settings identical to RepMLPNets; the ResMLP-S12 is trained with the official code and the same hyper-parameters as the reported 120-epoch result [33] except that we use a smaller batch size due to limited resources (but our reproduced accuracy is 2.7% higher than that reported by the paper [33]).

Compared to the other ConvNets and MLP models, we make the following observations. **1)** With fair settings, RepMLPNet shows superiority over the other MLPs: RepMLPNet-T256 outperforms MLP-Mixer (our impl, 256×256 inputs) by 0.49% in the accuracy while the FLOPs of the former is only 1/4 of the latter. **2)** With simple training methods, ResMLP and MLP-Mixer signif-

Table 2. Results on ImageNet classification. The throughput is tested on the same 2080Ti GPU with a batch size of 128.

Model	Input resolution	Train epochs	Top-1 acc	FLOPs (B)	Params (M)	Throughput
RegNetX-3.2GF [30]	224	120	78.4	3.2	15.2	988
ResNet-50 [17]	224	120	77.8	4.1	25.5	1007
ResNeXt-50 [38]	224	120	78.8	4.2	24.9	756
RegNetX-6.4GF [30]	224	120	79.6	6.4	26.2	589
ResNet-101 [17]	224	120	79.4	8.1	44.4	606
ResNeXt-101 [38]	224	120	80.2	8.0	44.1	450
RepMLPNet-T224	224	**100**	**76.4**	2.79	38.3	1709
ResMLP-S12 (our impl)	224	120	70.4	3.0	15.4	1895
ResMLP-S12 [33]	224	120	67.7	3.0	15.4	-
ResMLP-S12 + DeiT-train [33]	224	400	76.6	3.0	15.4	-
RepMLPNet-T256	256	**100**	**77.5**	4.24	58.7	1374
ResMLP-S24 + DeiT-train [33]	224	400	79.4	6.0	30.0	961
RepMLPNet-B224	224	**100**	**80.1**	6.65	68.2	816
RepMLPNet-D256	256	**100**	**80.8**	8.61	86.9	715
RepMLPNet-B256	256	**100**	**81.0**	9.61	96.5	708
S^2-MLP-deep [39]	224	300	80.7	10.5	51	-
RepMLPNet-L256	256	**100**	**81.8**	11.5	117.7	588
MLP-Mixer-B/16 [32]	224	300	76.4	12.6	59	-
S^2-MLP-wide [39]	224	300	80.0	14.0	71	-
MLP-Mixer-B/16 (our impl)	224	**100**	76.7	12.6	59.8	632
gMLP-B [22]	224	300	81.6	15.8	73	-
MLP-Mixer-B/16 (our impl)	256	**100**	77.0	16.4	60.4	578
ResMLP-B24 + DeiT-train [33]	224	400	81.0	23.0	115.7	-

icantly degrade: the accuracy of ResMLP-S12 drops by 8.9% (76.6% $\rightarrow$ 67.7%) without the 300-epoch DeiT-style training. **3)** RepMLPNet with 100-epoch training delivers a favorable accuracy-efficiency trade-off: RepMLPNet-B256 matches the accuracy of ResMLP-B24 without DeiT-style distillation, consumes 1/4 training epochs, has only 40% FLOPs and fewer parameters. **4)** With the comparable FLOPs, MLPs are faster than ConvNets. For examples, RepMLP-S12 and RegNetX-3.2GF have comparable FLOPs, while the former runs almost 2$\times$ as fast as the latter; RepMLPNet-B224 has lower FLOPs than ResNeXt-101 (6.65B v.s. 8.0B) but runs much faster. This discovery suggests that MLPs are promising as high-throughput inference models.

There are two key designs in RepMLP Block: the Global Perceptron and the set-sharing linear mapping of FC3. We then study their effects by enlarging the number of share-sets $\mathbf{S}$ or ablating the Global Perceptron and observing the performance as well as the model size. Fig. 3 shows that Global Perceptron only adds negligible parameters and FLOPs (0.5M) but improves the accuracy by around 1%, which is expected as the Local Perceptron and Channel Perceptron do not communicate information across channels, which is compensated by Global Perceptron. By enlarging $\mathbf{S}$, fewer channels will be sharing the same set of mapping parameters, resulting in a significant performance gain without any extra FLOPs. Therefore, for the application scenarios where the speed-accuracy trade-off is the primary concern while the model size is not (*e.g.*, in high-power computing centers), we may increase $\mathbf{S}$ for higher accuracy.

Table 3. Ablation studies on RepMLP-T224.

$\mathbf{S}$	Global Perceptron	Top-1 acc	FLOPs	Params
$[1, 4, 16, 128]$		75.78	2.7B	37.8M
$[1, 4, 16, 128]$	✓	76.48	+0.5M	38.3M
$[2, 8, 32, 256]$		75.94	2.7B	66.7M
$[2, 8, 32, 256]$	✓	77.19	+0.5M	67.2M

5.2. Locality Injection Matters

Before ImageNet, we first also conduct a series of ablation studies on CIFAR-100 [19] to verify the effectiveness of Locality Injection. Specifically, we build a small RepMLPNet with two stages, $\mathbf{B} = [6, 6]$, $\mathbf{S} = [8, 32]$, $C = 128$. As another benchmark model, we scale down ResMLP-12 by reducing the channel dimensions. Besides, we change the downsampling ratio at very beginning of all the models to 2×2, so that the embedding dimension becomes 16×16, which is close to the case of ResMLP designed for ImageNet (14×14). All the structural hyper-parameters are casually set since we do not intend to chase the state-of-the-art on such a small dataset.

For the ResMLP, we add 1×1 and 3×3 branches parallel to the spatial aggregation layer. Of note is that the spatial aggregation in ResMLP and MLP-Mixer is implemented by first transposing the input features, linearly mapping, and transposing back, which is equivalent to our set-sharing FC with only one share-set (*i.e.*, all the channels have the same set of $(hw)^2$ parameters). In this case, all the conv layers

Table 4. Studies on the effect of the Local Perceptron. The throughput is tested on the same 2080Ti GPU with a batch size of 128 and measured in examples/second. Of note is that the throughput is observably reduced by adding the conv layers with negligible FLOPs.

Dataset	Model	1×1 conv	3×3 conv	Top-1 acc	FLOPs	Params	Throughput
CIFAR-100	ResMLP-12			55.58	1812M	7.1M	2561
	ResMLP-12	✓		57.96	+786K	+60	2093
	ResMLP-12		✓	63.76	+7077K	+156	1858
	ResMLP-12	✓	✓	65.09	+7864K	+216	1619
CIFAR-100	RepMLPNet			59.07	468M	8.3M	6273
	RepMLPNet	✓		60.73	+294K	+720	5872
	RepMLPNet		✓	65.36	+2654K	+2640	5721
	RepMLPNet	✓	✓	67.43	+2949K	+3360	5328
ImageNet	RepMLPNet-T224			74.33	2.79B	38.3M	1709
	RepMLPNet-T224	✓	✓	76.48	+10M	+20K	1354
	RepMLPNet-D256			78.58	8.61B	86.94M	715
	RepMLPNet-D256	✓	✓	80.88	+26M	+60k	570

should have $s = 1$ accordingly, which means a DW conv with one input channel and one output channel. Consequently, adding a 1×1 conv introduces only five parameters (one for the $1\times1\times1\times1$ kernel and four for the single-channel BN layer including the moving mean, moving variance, scaling factor and bias), so that the whole model has only 5×12=60 extra parameters. Similarly, adding a 3×3 layer brings only $(3 \times 3 + 4) \times 12 = 156$ parameters. However, though the extra parameters and FLOPs are negligible, the speed is observably slowed down due to the reshaping operations and the reduction of degree of parallelism [26], which highlights the significance of merging the conv layers into the FC. For the RepMLPNet, the extra parameters brought by adding a 3×3 conv is $(3 \times 3 + 4)s$ due to the set-sharing mechanism.

All the ResMLPs and RepMLPNets on CIFAR-100 are trained with the same learning rate schedule and weight decay as described before, a batch size of 128 on a single GPU, and the standard data augmentation: padding to 40×40 and randomly cropping to 32×32. Predictably, the performance is not satisfactory since CIFAR-100 is too small for the FC layer to learn the inductive bias from the data. This discovery is consistent with the concurrent works [32, 33] which highlight that MLPs show inferior performance on small-scale datasets. However, adding the conv branches only during training significantly boost the accuracy even though they are eventually merged into the FC kernel, suggesting that the local priors injected do have a positive impact. Impressively, though the ResMLP has only 216 extra parameters, the accuracy raises by 9.51%. Then we observe a similar phenomenon on RepMLPNet. We then experiment with RepMLP-T224/D256 on ImageNet. With the Local Perceptron removed, the accuracy decreases by 2.15% and 2.30, respectively. The gap is narrower compared to the results on CIFAR, which is expected because ImageNet is significantly larger, allowing the model to learn some inductive bias from data [32]).

In summary, as Locality Injection works on different models and datasets, we conclude that it is a universal tool for vision MLPs.

5.3. Semantic Segmentations

The hierarchical design of RepMLPNet qualifies it as a backbone model for the off-the-shelf downstream framework that requires to combine the feature maps of different levels, *e.g.*, UperNet [37]. However, transferring an ImageNet-pretrained MLP to the downstream task is challenging. Taking Cityscapes [5] semantic segmentation as the example, we reckon there are two primary obstacles for using an MLP backbone. **1)** The backbone is usually trained with low resolution (*e.g.*, 256×256 on ImageNet) then transferred to the high-resolution task (*e.g.*, 1024×2048 of Cityscapes). However, the parameter count of MLP is coupled with the input resolution (by our specific definition of "MLP"), making the transfer difficult. **2)** The resolution for training (512×1024 of Cityscapes) and testing (1024×2048) may not be the same, so the backbone has to adapt to a variable resolution.

In brief, our solution is to split the inputs into non-overlapping patches, feed the patches into the backbone, restore the output patches to form the feature maps, which are then fed into the downstream frameworks. For example, a the first RepMLP Block of RepMLPNet-D256 works with 64×64 inputs because the FC kernel is $(64^2, 64^2)$. On Cityscapes, we can split the feature map into several 64×64 patches and feed the patches into the RepMLP Block. However, doing so breaks the correlations among patches, hence hinders a global understanding of the semantic information. Accordingly, we propose to replace the embedding (2×2 conv) layers by regular conv (3×3 conv) layers to communicate information across the patch edges.

As the first attempt to use an MLP as the backbone for Cityscapes semantic segmentation, the results are promising: with RepMLPNet-D256 + UperNet, we achieve comparable results with ResNet. The results and analysis are presented in the supplementary materials.

6. Limitations and Conclusions

This paper proposes a re-parameterization methodology to inject locality into FC layers, a novel MLP-style block, and a hierarchical MLP architecture. The proposed RepMLPNet is favorable compared to several concurrently proposed MLP architectures in terms of the accuracy-efficiency trade-off and the training costs.

However, as an MLP, RepMLPNet has several noticeable common weaknesses. 1) Similar to the Vision Transformers, MLPs are easy to overfit, hence requiring strong data augmentation and regularization techniques to reach a satisfactory level of validation accuracy. 2) On the low-power devices like mobile phones, the model size of MLPs may be an obstacle. 3) Though the results of our first attempt to use MLP backbone for semantic segmentation are promising, we observe no superiority over the traditional ConvNets.

References

[1] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006. 4

[2] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. Cyclemlp: A mlp-like architecture for dense prediction. *arXiv preprint arXiv:2107.10224*, 2021. 2

[3] Minsik Cho and Daniel Brand. Mec: memory-efficient convolution for deep neural network. In *International Conference on Machine Learning*, pages 815–824. PMLR, 2017. 4

[4] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020. 11

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 3213–3223. IEEE Computer Society, 2016. 8

[6] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 6

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 1, 6

[8] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019. 3

[9] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616, 2019. 3

[10] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1911–1920, 2019. 3

[11] Xiaohan Ding, Tianxiang Hao, Jungong Han, Yuchen Guo, and Guiguang Ding. Manipulating identical filter redundancy for efficient pruning on deep and complicated cnn. *arXiv preprint arXiv:2107.14444*, 2021. 3

[12] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021. 3

[13] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10886–10895, 2021. 3

[14] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021. 2, 3

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 2

[16] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 4

[17] 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, 7

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 2, 3

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 7

[20] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016. 4

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 3

[22] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021. 1, 2, 3, 6, 7

[23] 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. *arXiv preprint arXiv:2103.14030*, 2021. 2, 6

[24] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In Yoshua Bengio and Yann LeCun, editors, *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 3

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[26] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 2, 8

[27] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013. 4

[28] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 4

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. 4

[30] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 6, 7

[31] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 6

[32] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021. 1, 2, 6, 7, 8

[33] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021. 1, 2, 3, 6, 7, 8

[34] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021. 1, 2, 6

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. 1

[36] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018. 1

[37] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 418–434, 2018. 1, 6, 8, 11

[38] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 6, 7

[39] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S2-mlp: Spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2106.07477*, 2021. 1, 6, 7

[40] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019. 6

[41] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 6

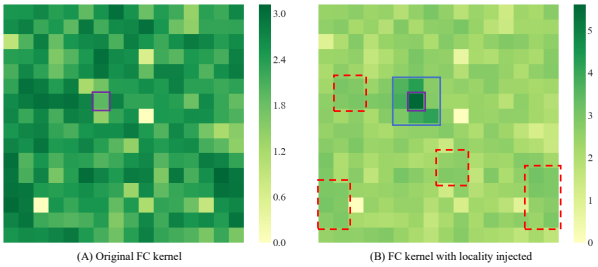


Figure 3. FC kernel before and after Locality Injection.

Appendix A: Visualizing Locality Injection

We demonstrate the locality is injected into the FC kernel by showing the kernel weights.

Specifically, we visualize the weights of FC3 kernel sampled from the 10th RepMLP Block of the 3rd stage of RepMLPNet-D256 trained on ImageNet. We reshape the kernel into $\bar{\mathrm{W}}(s,h,w,1,h,w)$, which is $(16,16,16,1,16,16)$, then sample the weights related to the first input channel and the (6,6) point (marked by a purple square) on the first output channel, which is $\bar{\mathrm{W}}_{0,6,6,0,:,:}$. Then we take the absolute value, normalize by the minimum of the whole matrix, and take the logarithm for the better readability. A point with darker color indicates the FC considers the corresponding position on the input channel more related to the output point at (6,6).

Fig. 3 shows that the original kernel has no locality as the marked point and the neighborhood have no larger values than the others.

Then we merge the parallel conv layers into the kernel via Locality Injection and the resultant kernel has larger values around the marked point, suggesting that the model focuses more on the neighborhood, which is expected. Besides, the kernel's capacity to model the long-range dependencies is not lost as some points (marked by red dashed rectangles) outside the largest conv kernel (3×3 in this case, marked by a blue square) still have larger values than some points inside.

Appendix B: Details of Semantic Segmentation

We solve the problem of using MLP as the backbone of a semantic segmentation framework (*e.g.*, UperNet) by **1)** the hierarchical design, **2)** splitting feature maps into non-overlapping patches and **3)** communications between patches.

Fig. 4 shows an example of RepMLPNet + UperNet.

1) UperNet requires feauture maps of 4 different levels, which fits our hierarchical architecture. **2)** Since RepMLP Block works with a fixed input feature map size, we split the feature maps into non-overlapping patches each with the required size. **3)** The original 2× embedding cannot realize inter-patch communication, so we replace it with 3×3 conv. To reduce the computational cost of 3×3 conv, we decompose it into a 1×1 conv for expanding the channels and a 3×3 stride-2 depthwise conv for downsampling. Fig. 5 shows the difference between a 2× embedding and a 3×3 conv.

Table 5. Results on Cityscapes val set. By replacing the 3×3 downsampling layers with 5×5, the mIoU further increases.

Backbone	FLOPs	mIoU
ResNet-101	2049.82G	79.03
RepMLPNet-D256	1960.01G	76.27
RepMLPNet-D256 (conv5)	1960.16G	77.12

Interestingly, as the input is split into non-overlapping patches, one may expect that the predictions would be less accurate at the boundaries of patches, but we observe no such phenomenon. Fig. 6 shows that the predictions are as good at the boundaries as the internal pixels within patches. This discovery suggests that the dependencies across patches have been well established and that the representational capacity of MLP is strong enough for such a dense prediction task.

As the first attempt to use an MLP backbone for semantic segmentation, RepMLPNet delivers a promising result. Table. 5 shows that the performance of RepLKNet is comparable to traditional ConvNet. By further replacing the 3×3 downsampling layer by 5×5, the mIoU improves with negligible extra FLOPs, which is expected as a larger convolution enables better communications among patches. Specifically, we use the open-sourced implementation of UperNet [37] in MMSegmentation [4] and the 80K-iteration training schedule.

We hope our results spark further research on the application of MLP on downstream tasks.

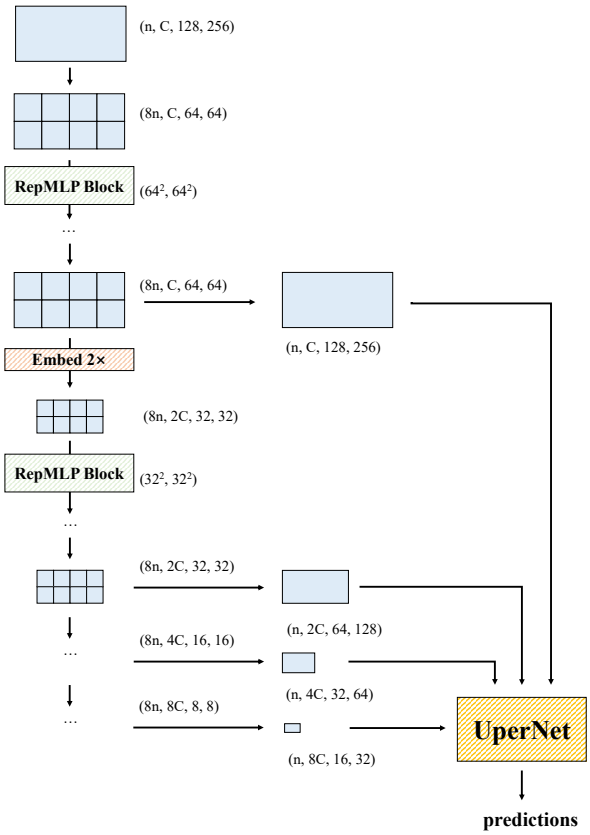
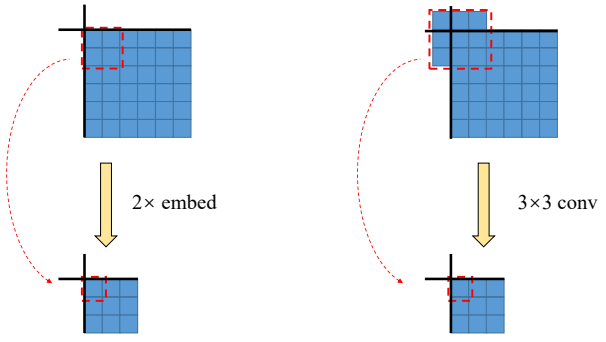


Figure 4. An example of using RepMLPNet as the backbone of UperNet. After 4× downsampling on the 512×1024 inputs, the feature map size becomes 128×256. Then we split the feature maps into 2×4 non-overlapping patches, each of 64×64, because the first RepMLP Block maps inputs of 64×64 into 64×64 (*i.e.*, the FC kernel is (64^2,64^2)). Similarly, the outputs of the four stages are reshaped back and fed into the UperNet.



(A) Downsample with 2× embed (B) Downsample with 3×3 conv

Figure 5. The difference between 2× embedding and 3×3 conv is that the latter realizes communications across patch edges. In this figure, a square denotes a pixel on a feature map and the thick lines denote the boundaries of patches. We take the upper left corner of a patch as an example

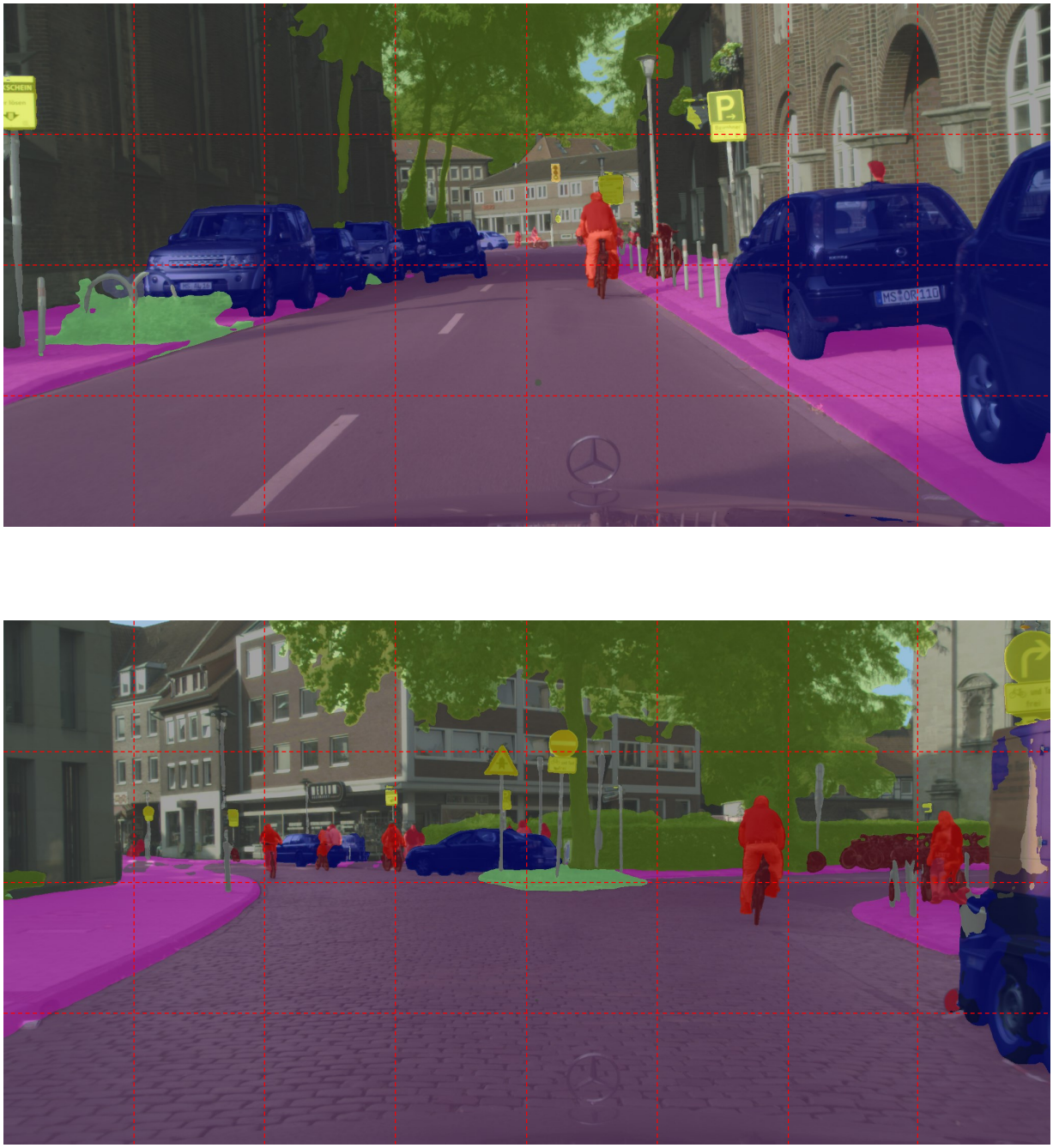


Figure 6. The predictions at the boundaries of patches are no observably worse. We show two images from the Cityscapes validation set as examples. As the test resolution is 1024×2048, the input to the first RepMLP Block is 256×512 (after the beginning 4× downsampling), so that the input is split into 32 non-overlapping patches and then fed into RepMLP Blocks. We use red dashed lines to denote the boundaries of patches and it is observed that the predictions at the boundaries are almost as good as the other parts.