ConvMLP: Hierarchical Convolutional MLPs for Vision

Abstract

MLP-based architectures, which consist of a sequence of consecutive multi-layer perceptron blocks, have recently been found to reach comparable results to convolutional and transformer-based methods. However, most adopt spa- tial MLPs which take fixed dimension inputs, therefore mak- ing it difficult to apply them to downstream tasks, such as object detection and semantic segmentation. More- over, single-stage designs further limit performance in other computer vision tasks and fully connected layers bear heavy computation. To tackle these problems, we propose Con- vMLP: a hierarchical Convolutional MLP for visual recog- nition, which is a light-weight, stage-wise, co-design of convolution layers, and MLPs. In particular, ConvMLP- S achieves 76.8% top-1 accuracy on ImageNet-1k with 9M parameters and 2.4 GMACs (15% and 19% of MLP-Mixer-B/16, respectively). Experiments on object detection and semantic segmentation further show that visual repre- sentation learned by ConvMLP can be seamlessly trans- ferred and achieve competitive results with fewer param- eters.

Keywords: MLP, ConvMLP, light-weight, stage-wise

Link to paper:https://arxiv.org/abs/2109.04454
Project Link :https://github.com/SHI-Labs/Convolutional-MLPs

1 Introduction

MLP-based models show that simple feedforward neural networks can compete with operators such as convolution and attention for image classification. However, using MLPs to encode spatial information requires fixing the dimensionality of the input, which makes it difficult to deploy on downstream computer vision tasks, which typically require input sizes of arbitrary resolution. In addition, the single-stage design after ViT [2] may limit the performance of object detection and semantic segmentation. Large sequential MLPs also bring heavy computational burdens and more parameters with high hidden layer dimensions.MLP-Mixer [12] only slightly outperforms large variants of ViTBase [14] by more than twice as much computationally and by more than twice as much as ViTBase. Similarly, ResMLP has more than 30% more parameters and complexity compared to a transformer-based model with similar performance.

Based on these observations, we propose ConvMLP: a hierarchical convolutional MLP backbone for visual recognition that is a combination of convolutional and MLP layers that can be seamlessly used for downstream tasks such as target detection and segmentation. As shown in Figure 1, in order to remove

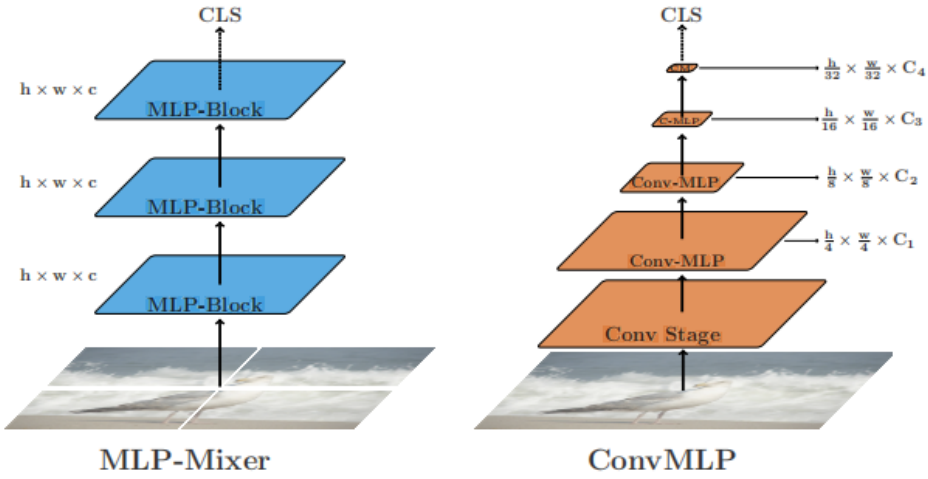

Figure 1. Compare the MLP mixer with ConvMLP. ConvMLP uses simple hierarchical multi-stage co-design with convolution and MLP.

the restriction on input dimensions in other MLP-like frameworks, we first replace all spatial MLPs with channel MLPs connected across channels and build a pure MLP baseline model. To compensate for the interaction of spatial information, we add a lightweight convolutional stage on top of the remaining MLP stages and use the convolutional layer for downsampling. In addition, to enhance the spatial connectivity in the MLP stage, we add a simple 3×3 deep convolution between the two channel MLPs in each MLP block, and thus refer to it as a Conv-MLP block. This synergistic design of the convolutional and MLP layers builds a prototype of the ConvMLP model for image classification. To make ConvMLP scalable, we extend the ConvMLP model by scaling the depth and width of the convolution and Conv-MLP stages. It achieves competitive performance on ImageNet-1 k compared to recent MLP-based models.

2 Related works

2.1 Convolutional Methods

classification has been dominated by convolutional neural networks for almost a decade, since the rise of AlexNet [7] , which introduced a convolutional neural network for image classification, and won the 2012 ILSRVC. Following that, VGGNet [11] proposed larger and deeper network for better performance. ResNet [3] introduced skip connections to allow training even deeper networks, and DenseNet [5] proposed densely connected convolution layers. In the meantime, researchers explored smaller and more light-weight models that would be deployable to mobile devices. MobileNet consisted of depth-wise and point-wise convolutions, which reduced the number of parameters and computation required. ShuffleNet [10] found channel shuffling to be effective, and EfficientNet further employs model scaling to width, depth and resolution for better model scalability.

2.2 Transformer-based Methods

Transformer [14] was proposed for machine translation and has been widely adopted in most natural language processing. Recently, researchers in computer vision area adopt transformer to image classification. They propose ViT [2] that reshapes image to patches for feature extraction by transformer encoder, which achieves comparable results to CNN-based models. DeiT [13] further employs more data augmentation and makes ViT comparable to CNN-based model without ImageNet-22k or JFT-300M pretraining. DeiT also proposes an attention-based distillation method, which is used for student-teacher training, leading to even better performance. PVT [15] proposes feature pyramids for vi sion transformers, making them more compatible for downstream tasks. Swin Transformer [9] uses patch-level multiheaded attention and stage-wise design, which also increase transferability to downstream tasks. Shuffle Swin Transformer [6] proposes shuffle multi-headed attention to augment spatial connection between windows. CCT proposes a convolutional tokenizer and compact vision transformers, leading to better performance on smaller datasets training from scratch, with fewer parameters compared with ViT. TransCNN [8] also proposes a co-design of convolutions and multi-headed attention to learn hierarchical representations.

2.3 MLP-based Methods

MLP-Mixer [12] was recently proposed as a large scale image classifiers that was neither convolutional nor transformer-based. At its core, it consisted of basic matrix multiplications, data layout changes and scalar nonlinearities. ResMLP [7] followed a ResNet-like structure with MLP-based blocks instead of convolutional ones. Following that, gMLP proposed a Spatial Gating Unit to process spatial features. S2-MLP adopts shifted spatial feature maps to augment information communication. ViP employs linear projection on the height, width and channel dimension separately. All these methods have MLPs on fixed spatial dimensions which make it hard to be used in downstream tasks since the dimensions of spatial MLPs are fixed. Cycle MLP and AS-MLP are concurrent works. The former replaces the spatial MLPs with cycle MLP layers and the latter with axial shifted MLPs, which make the model more flexible for varying inputs sizes. They reach competitive results on both image classification and other downstream tasks. Hire-MLP is an- other concurrent work that uses Hire-MLP blocks to learn hierarchical representations and achieves comparable result to transformer-based model on ImageNet.

3 ConvMLP

3.1 Overall Design

The general framework of ConvMLP is shown in Figure 2. Unlike other MLP-based models, we use a convolutional tagger to extract the initial feature map F1 (H 4 $\times$ W 4 $\times$ C1 dimensions). To reduce computation and improve spatial connectivity, we perform tokenisation using a purely convolutional stage to produce feature map F2 (H 8 $\times$ W 8 $\times$ C2 dimensional). We then place 3 Conv-MLP stages to generate 2 feature maps F3 and F4 (H 16 $\times$ W 16 $\times$ C3 and H 32 $\times$ W 32 $\times$ C4 dimensions,

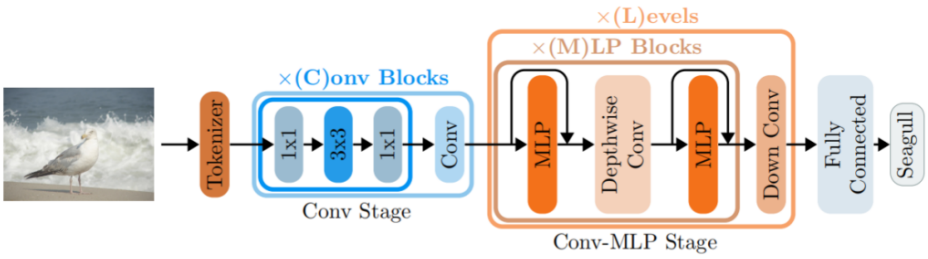

Figure 2. Overview of ConvMLP framework. The Conv Stage consists ofC convolutional blocks with 1×1 and 3×3 kernel sizes.

respectively). Each Conv-MLP stage consists of multiple Conv-MLP blocks and each Conv-MLP block has a channel MLP followed by a deep convolutional layer followed by another channel MLP.Similar to previous works, we include residual concatenation and layer normalisation applied to the inputs in the blocks. Each channel MLP consists of two fully connected layers with GeLU [4] activation and dropout. we then apply global average pooling to the output feature map F4 and send it to the classification header.

3.2 Convolutional Tokenizer

As stated, we replace the original patch tokenizer with a convolutional tokenizer. It includes three convolutional blocks, each consisting of a 3x3 convolution, batch nor- malization and ReLU activation. The tokenizer is also appended with a max pooling layer.

3.3 Convolution Stage

In order to augment spatial connections, we adopt a fully-convolutional first stage. It consists of multiple blocks, where each block is comprised of two 1x1 convolution layers with a 3x3 convolution in between.

3.4 Conv-MLP Stage

To reduce the constraints on the input dimensions, we replace all spatial MLPs with channel MLPs.Since channel MLPs only share weights between channels and lack spatial interactions, we compensate for it by adding convolutional layers in the early stage, downsampling, and MLP blocks. In the baseline model we follow Swin Transformer [9] and to add adjacent spatial crossings we replace patch merging with a 3x 3 convolutional layer at step 2. convolution in MLP block We further add depth direction convolutional layer between two channel MLPs in an MLP block and name it as Conv-MLP block. It is a 3x 3 convolutional layer with the same channels as the two channel MLPs, which was also used in the recent Shuffle Swin Transformer [6] to enhance neighbourhood window connections. It compensates for the lack of removing spatial MLPs and improves performance more substantially with the introduction of fewer parameters.

4 Implementation details

4.1 Comparing with the released source codes

Some of the main code in the project includes these classes, ConvMLP, convmlp, etc. Among the key to this model are these steps ConvTokenizer, ConvStage and Conv-MLP Stage.here the focus is on analyzing the code reproduction of these three classes.

Replication of ConvTokenizer

One of the surprises I got from the ConvMLP work was telling me in Related Work that CCT had proposed the Convolutional Tokenizer to replace Patch Embedding and improve performance.

The concrete implementation of Convolutional Tokenizer in ConvMLP is very simple: for example, a 244*244*3 natural image as input, after three $3 \times 3 \times 3$ convolutions, BN and ReLU, and finally after a pooling layer it ends. It is worth noting that the first convolutional layer has stride = 2, and finally there is a maximum pooling with stride = 2 and kernel size = 3.

```
class ConvTokenizer(nn.Module):
  def __init__(self, embedding_dim=64):
        super(ConvTokenizer, self).__init__()
        self.block = nn.Sequential(
                nn.Conv2d(3,
                embedding_dim // 2,
                 kernel_size=(3, 3),
                  stride=(2, 2),
                  padding=(1, 1),
                   bias=False),
                nn.BatchNorm2d(embedding_dim // 2),
                nn.ReLU(inplace=True),
                nn.Conv2d(embedding_dim // 2,
                 embedding_dim // 2, kernel_size=(3, 3),
                  stride=(1, 1),
                  padding=(1, 1),
                  bias=False),
                nn.BatchNorm2d(embedding_dim // 2),
                nn.ReLU(inplace=True),
                nn.Conv2d(embedding_dim // 2,
                 embedding_dim,
                  kernel_size=(3, 3),
                  stride=(1, 1),
                   padding=(1, 1),
                    bias=False),
```

```
                    nn.BatchNorm2d(embedding_dim),
                    nn.ReLU(inplace=True),
                    nn.MaxPool2d(kernel_size=(3, 3),
                     stride=(2, 2),
                      padding=(1, 1),
                      dilation=(1, 1))
            )
```

Replication of ConvStage

In order to increase the interaction of information in the space, the authors used a fully convolutional stage. It consists of multiple blocks, where each block consists of two 1×1 convolutional layers with a 3×3 convolutional layer in the center. And residuals are performed, which are actually modeled after ResNet50. Finally a 3×3 convolution with stride $= 2$ is used to achieve downsampling.

```
class ConvStage(Module):
        def __init__(self,
                    num_blocks=2,
                    embedding_dim_in=64,
                    hidden_dim=128,
                    embedding_dim_out=128):
                    super(ConvStage, self).__init__()
                    self.conv_blocks = ModuleList()
                    for i in range(num_blocks):
                    block = Sequential(
                    Conv2d(embedding_dim_in,
                    hidden_dim,
                    kernel_size=(1, 1),
                    stride=(1, 1),
                    padding=(0, 0),
                    bias=False),
                    BatchNorm2d(hidden_dim),
                    ReLU(inplace=True),
                    Conv2d(hidden_dim, hidden_dim,
                    kernel_size=(3, 3),
                    stride=(1, 1),
                     padding=(1, 1),
                     bias=False),
                    BatchNorm2d(hidden_dim),
                    ReLU(inplace=True),
                    Conv2d(hidden_dim,
                    embedding_dim_in,
```

```
                    kernel_size=(1, 1),
                    stride=(1, 1),
                    padding=(0, 0),
                    bias=False),
                  BatchNorm2d(embedding_dim_in),
                  ReLU(inplace=True)
                  )
                  self.conv_blocks.append(block)
                  self.downsample = Conv2d(embedding_dim_in,
                  embedding_dim_out,
                  kernel_size=(3, 3),
                  stride=(2, 2),
                  padding=(1, 1))
```

Replication of Conv-MLP Stage

In order to reduce the constraints on the input dimensions, the authors replace all the Token-mixing MLPs with Channel-mixing MLPs (1×1 convolution).However, this leads to a lack of spatially informative interactions, so the authors compensate for the lack of spatial interactions by adding convolutional layers for local informative interactions.

Each Channel-mixing MLP is in fact first normalized by LN, and then two fully connected layers in the channel direction, GELU activation function. This can be regarded as 1×1 convolution. The middle convolutional layer then uses a 3×3 Depthwise Conv, which compensates for the interaction of spatial information and brings in very few parameters. Note: ConvMixer is honest enough to say that its 1×1 plus 3×3 convolution is a convolutional neural network, and that the Block is a ConvBlock, not that it is fully connected.

In addition, the authors follow the design approach used in Swin Transformer to downsample the feature maps using a patch merging method based on linear layers. The difference is that the authors replace the patch merger (2×2 convolution with a step size of 2) with a 3×3 convolutional layer with a step size of 2, which allows for overlapping spatial information when downsampling. This improves the classification accuracy while introducing very few parameters.

```
class Mlp(nn.Module):
        def __init__(self,
                  embedding_dim_in, hidden_dim=None,
                   embedding_dim_out=None,
                   activation=nn.GELU):
                  super().__init__()
                  hidden_dim = hidden_dim or embedding_dim_in
                  embedding_dim_out = embedding_dim_out or embedding_dim_in
                  self.fc1 = nn.Linear(embedding_dim_in, hidden_dim)
                  self.act = activation()
```

```
                self.fc2 = nn.Linear(hidden_dim, embedding_dim_out)

        def forward(self, x):
                return self.fc2(self.act(self.fc1(x)))


class ConvMLPStage(nn.Module):
        def __init__(self, embedding_dim, dim_feedforward=2048,
        stochastic_depth_rate=0.1):
                super(ConvMLPStage, self).__init__()
                self.norm1 = nn.LayerNorm(embedding_dim)
                self.channel_mlp1 = Mlp(
                embedding_dim_in=embedding_dim,
                hidden_dim=dim_feedforward)
                self.norm2 = nn.LayerNorm(embedding_dim)
                self.connect = nn.Conv2d(embedding_dim,
                embedding_dim, kernel_size=(3, 3), stride=(1, 1),
         padding=(1, 1), groups=embedding_dim, bias=False)
                self.connect_norm = nn.LayerNorm(embedding_dim)
                self.channel_mlp2 = Mlp(embedding_dim_in=embedding_dim,
         hidden_dim=dim_feedforward)
                self.drop_path = DropPath(stochastic_depth_rate)
                        if stochastic_depth_rate > 0 else nn.Identity()

                def forward(self, src):
                src = src + self.drop_path(self.channel_mlp1(
                self.norm1(src)))
                src = self.connect(
                self.connect_norm(src).permute(0,3,1,2)).permute(0,2,3,1)
                src = src + self.drop_path(
                self.channel_mlp2(self.norm2(src)))
                return src


class ConvDownsample(nn.Module):
        def __init__(self, embedding_dim_in, embedding_dim_out):
                super().__init__()
                self.downsample = nn.Conv2d(embedding_dim_in,
                embedding_dim_out, kernel_size=(3, 3),
                stride=(2, 2), padding=(1, 1))
```

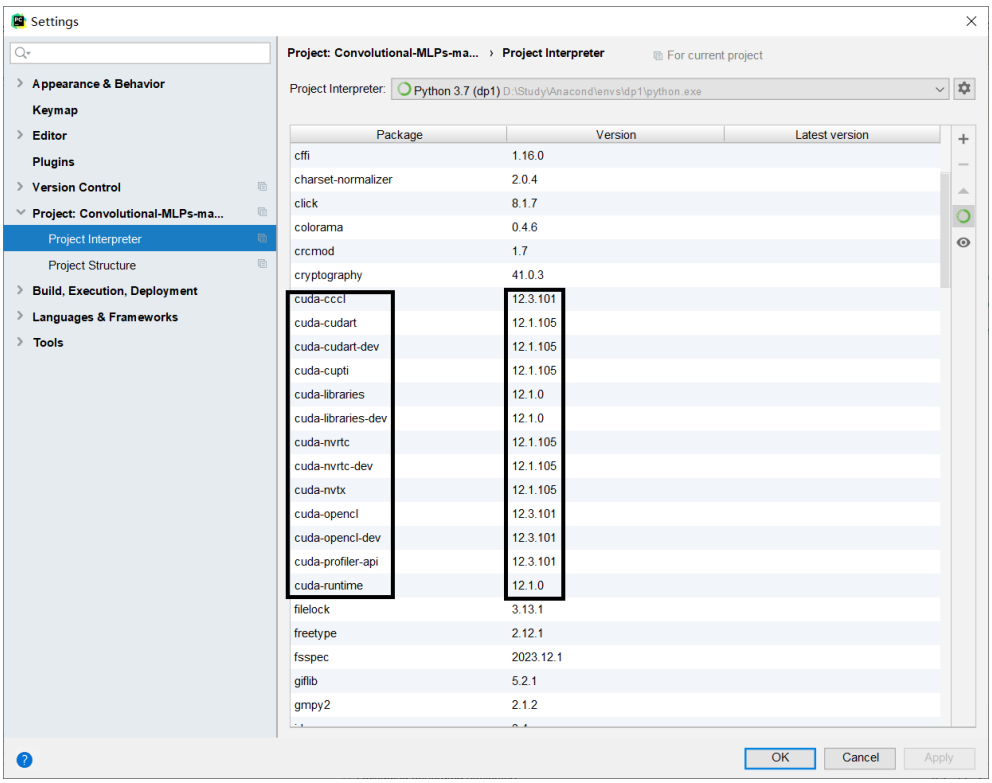

Figure 3. Some configuration of cuda

```
        def forward(self, x):
                x = x.permute(0, 3, 1, 2)
                x = self.downsample(x)
                return x.permute(0, 2, 3, 1)
```

4.2 Experimental environment setup

The environment in which the model is reproduced is based on window10, pycharm-gpu, python3.9. The configuration of the environment is shown below. The cuda used for GUP acceleration, as shown here Figure 3. The libraries needed for the project, such as pytorch, torchvision, timm, etc. are shown in the following Figure 4. The dataset used for your own replication is shown in Figure 5.

4.3 Main contributions

ImageNet-1k [7] contains 1.2M training images and 50k images on 1000 categories for evaluating performances of classifiers. We follow standard practice provided by timm toolbox. We use RandAugment [1] Mixup [17] , and CutMix [1] for data augmentation. AdamW is adopted as optimizer with momentum of 0.9 and weight decay of 0.05. The initial learning rate is 0.0005 with batch size of 128 on each GPU card. We use 8 NVIDIA RTX A6000 GPUs to train all models for 300 epochs and the total batch size is 1024. All other training settings and hyperparameters are adopted from Deit [13] for fair comparisons. For those results in ablation study, we train these models for 100 epochs with batch size 256 on each GPU and use 4 GPUs with learning rate at 0.001.

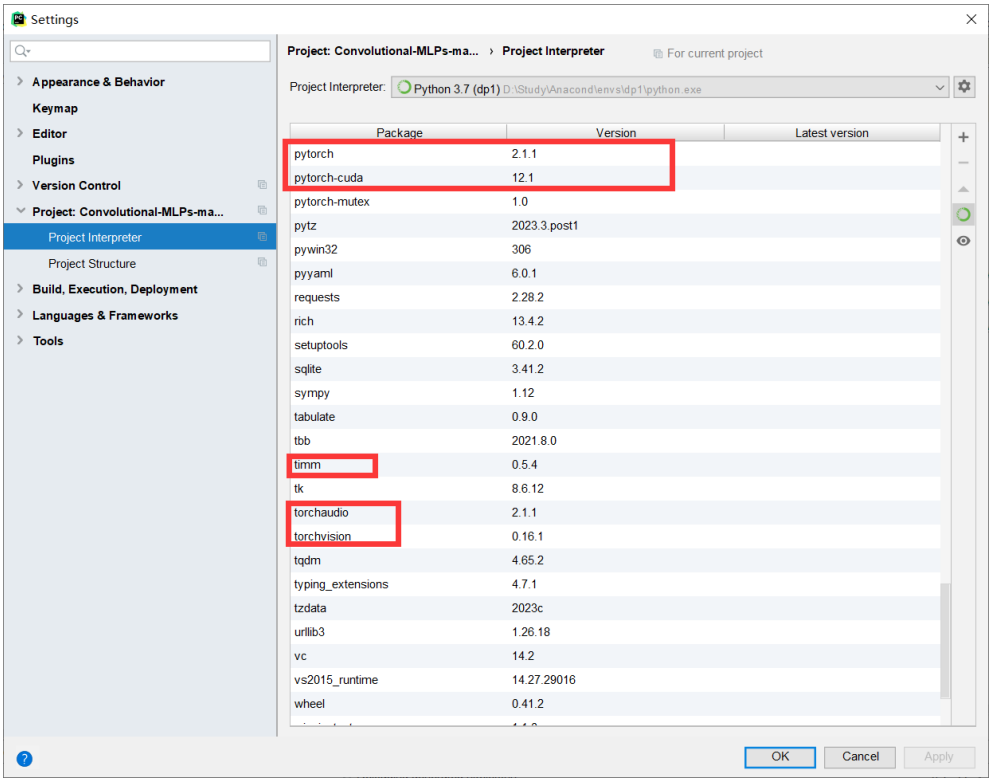

Figure 4. Main libraries required for the project

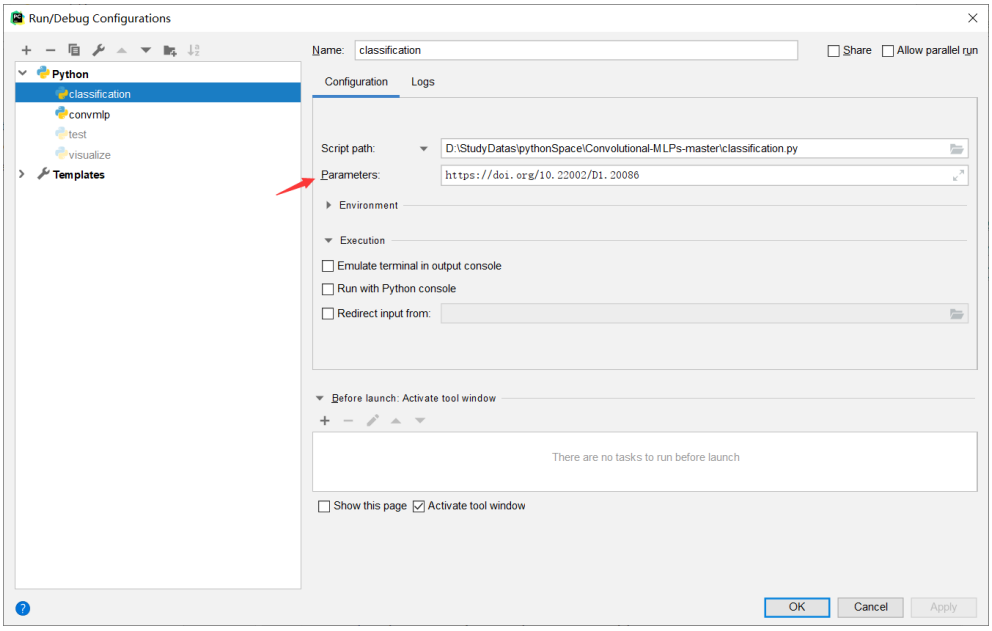

Figure 5. Number of experiments

In Table, we compare ConvMLP to other state-of- the-art image classification models on ImageNet-1k. We include Convolution-based, Transformer-based and MLP- based methods under different scales. We also present num- ber of parameters and GMACs of these models.

5 Results and analysis

The authors visualise the feature maps of ResNet 50, MLPMixer-B/16, Pure-MLP Baseline and ConvMLP-M at (1024, 1024) input size (MLP-Mixer-B/16 at (224, 224) due to the size constraints) in Figure 6 in order to analyse the visual representations learnt by these models for the differences, similar feature maps for transformer-based models are provided in T2 T-ViT [16] . We observe that the representations learnt by ConvMLP involve more underlying features, such as edges or textures, compared to ResNet, and more semantics compared to the Pure-MLP Baseline.

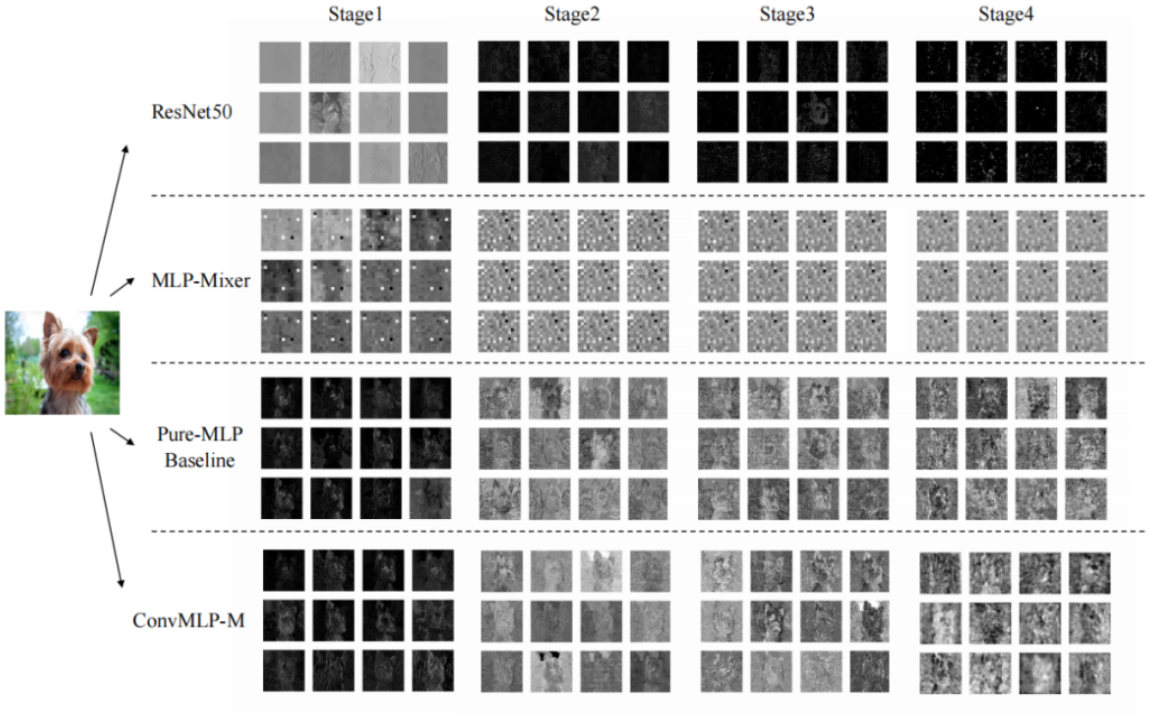

Figure 6. Visualization of feature maps in different stages of ResNet50, MLP-Mixer, Pure-MLP Baseline and ConvMLP-M. Visual representations learned by ConvMLP-M show both semantic and low-level information.

6 Conclusion and future work

This paper analyses the limitations of current MLP-based visual representation learning models: to address some of the problems of traditional MLPs, the authors propose ConvMLP: a hierarchical convolutional MLP that combines convolutional layers and MLPs for visual representation learning. The architecture can seamlessly pre-consider downstream networks such as RetinaNet, MaskR-CNN and semantic FPN. Experiments further show that the method can achieve competitive results on different

benchmarks with fewer parameters compared to other methods.

The main limitation of ConvMLP is that ImageNet performance grows slower with model size. We leave this to be explored in future work.

References

[1] 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.

[2] 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.

[3] 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.

[4] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415, 2016.

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

[6] Zilong Huang, Youcheng Ben, Guozhong Luo, Pei Cheng, Gang Yu, and Bin Fu. Shuffle transformer: Rethinking spatial shuffle for vision transformer. arXiv preprint arXiv:2106.03650, 2021.

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

[8] Yun Liu, Yu-Huan Wu, Guolei Sun, Le Zhang, Ajad Chhatkuli, and Luc Van Gool. Vision transformers with hierarchical attention. arXiv preprint arXiv:2106.03180, 2021.

[9] 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 Proceedings of the IEEE/CVF international conference on computer vision, pages 10012–10022, 2021.

[10] 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.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

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

[13] 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.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[15] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In Proceedings of the IEEE/CVF international conference on computer vision, pages 568–578, 2021.

[16] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In Proceedings of the IEEE/CVF international conference on computer vision, pages 558–567, 2021.

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