Inception Transformer

吴倍亮

摘要

最近的研究表明，Transformer 具有强大远程建模能力，但 Transformer 无法捕获捕捉高频信息 = 处理局部信息存在问题。为了解决这个问题，作者提出了一个新的和通用 Inception Transformer，简称 iFormer，可以有效地学习具有视觉中高频和低频信息的综合特征数据。具体来说，作者设计了一个 Inception Mixer 直接移植用于捕获高频信息的卷积和具有更强大的捕抓低频信息能力的 Transformer。与最近的混合框架不同，Inception 混合器通过通道拆分机制采用卷积/最大池路径和自注意力并行路径带来更高的效率，并将此命名为高频与低频混合器。同时考虑到底层 layers 在捕捉高频细节发挥更多作用，同时顶层应该更多地建模低频全局信息，作者进一步引入频率斜坡结构，即逐渐减小输入到高频卷积的比例并增加到低频 transformer 中的 MSA(Multi-head Attention)，它可以有效地平衡不同层的高频和低频分量。文章对 iFormer 在视觉任务上进行了一系列基准测试。

关键词：轻量化；Inception Mixer；斜坡结构

1 引言

Transformer[1]席卷了自然语言处理 (NLP) 领域，在许多 NLP 任务（例如机器翻译和问答）中实现了惊人的高性能。这在很大程度上归功于其强大的 Self-Attention 机制对数据中的长期依赖关系进行建模的能力。它的成功促使研究人员研究它对计算机视觉领域的适应，而 Vision Transformer(ViT)[2]是先驱。该架构直接继承自 NLP，但应用于以原始图像块作为输入的图像分类。后来，许多 ViT 变体被开发出来，以提高性能或扩展到更广泛的视觉任务，例如目标检测和分割。ViT 及其变体在视觉数据中具有很强的捕获低频的能力，主要包括场景或对象的全局形状和结构学习，但对于学习高频的能力不是很强，主要包括局部边缘和纹理。这可以直观地解释：Self-Attention 是 ViT 中用于在非重叠 patch tokens 之间交换信息的主要操作，也是一种全局操作，相对于高频局部信息 Self-Attention 更能捕获数据中低频的全局信息。低频提供有关视觉刺激的全局信息，而高频传达图像中的局部空间变化（例如，局部边缘/纹理）。因此，有必要开发一种新的 ViT 架构来捕获视觉数据中的高频信息和低频信息。而这篇文章则是视觉邻域一个较新较好能兼顾高低频信息与模型参数的工作。频谱分析如图 1所示：

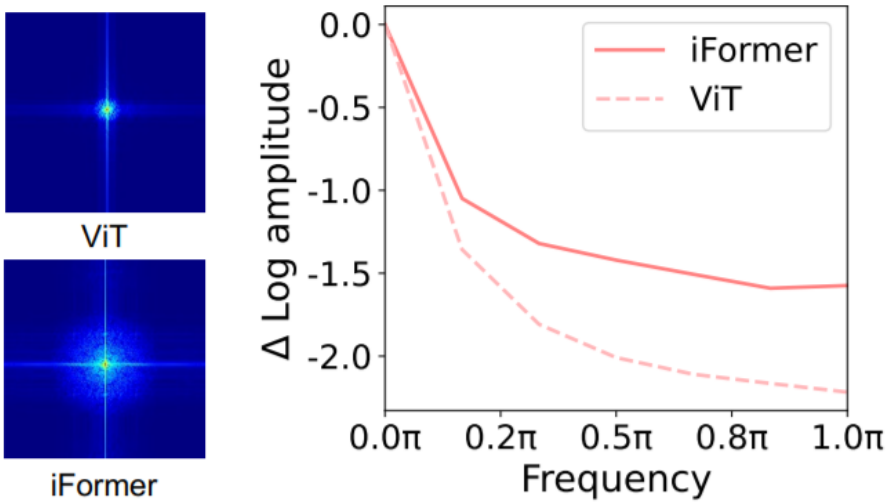

图 1: 频谱分析

2 相关工作

首先回顾一下 Vision Transformer。对于视觉任务，Transformer 首先将输入图像拆分为一系列 Token，每个 Patch Token 投影到具有更精简层的隐藏表示向量中，表示为 Patch Embedding，其中 N 是 Patch Token 的数量，C 表示特征的维度。然后，所有 Token 与位置嵌入相结合，并送到包含 Multi-Head Self-Attention(MSA) 和前馈网络 (FFN) 的 Transformer 层。

在 MSA 中，基于注意力的 Mixer 在所有 Patch Token 之间交换信息，因此它强烈关注聚合所有层的全局依赖关系。然而，全局信息的过度传播会加强低频表示并抑制高频。从图 1中的傅里叶谱的可视化可以看出，低频信息主导了 ViT 的表示。这实际上会损害 ViT 的性能，因为它可能会恶化高频成分，例如局部纹理，并削弱 ViT 的建模能力。实际上在视觉数据中，高频信息也具有判别力，可以使许多任务受益。

2.1 串行结构

CNN[3][4][5]是视觉任务的实际模型，因为它们具有出色的局部建模能力以及提取高频信息。有了这些优势，CNNs 得以串行或并行方式快速引入 Transformers。对于串行方法中，卷积应用于 Transformer 的不同位置。如在不同位置用一层重叠的卷积替换 Transformer 的 patch embedding 及 MSA，堆叠几层卷积层也作为模型的主干部分，虽然实验证明这有助于训练和获得更好的表现。然而，卷积和注意力的组合在一个连续的顺序意味着每一层只能处理高频或低频并忽略另一部分，不能使得每一层都能处理不同的频率。

2.2 平行结构

与串行方法相比，将注意力和卷积平行结合在一起的工作并不多。通过平行地引入卷积作为于注意力机制再合并两个分支的输出。但是研究发现这种结构中一些通道倾向于提取局部依赖性，而其他通道则用于建模全局信息，这表明当前并行机制处理所有通道是存在冗余的计算。因此，作者将通道拆分为高频和低频的分支，节省计算的同时能很好移植 CNN 捕抓高频信息和 Transformer 建模低频信息的优势。

3 本文方法

本文提出了一个 Inception mixer 将 CNN 提取高频表示的强大能力移植到 Transformer 中。其详细架构如图 2所示。Inception mixer 不是直接将图像输入到 MSA Mixer 中，而是首先沿通道维度分割输入特征，然后将分割后的分量分别输入到 high-frequency mixer 和 low-frequency mixer 中。这里的 high-frequency mixer 由一个最大池化操作和一个并行卷积操作组成，而 low-frequency mixer 由一个 Self-Attention 实现。

从技术上讲，给定输入特征图 X，沿通道维度将 X 分解为 Xl 和 Xh，然后将 Xl 和 Xh 分配给 high-frequency mixer 和 low-frequency mixer。

3.1 本文方法概述

整体结构如图 2所示，Backbone 有 4 个阶段如图 3所示，具有不同的通道和空间维度。对于每个 Block 定义了一个通道比以更好地平衡高频和低频分量，即 Ch/C 和 Cl/C，其中 Ch/C + Cl/C = 1。在建议的 Frequency ramp structure 中，Ch /C 由浅到深逐渐减小，而 Cl/C 逐渐增大。因此，通过灵活的频率斜坡结构，iFormer 可以有效地权衡所有层的高频和低频分量。

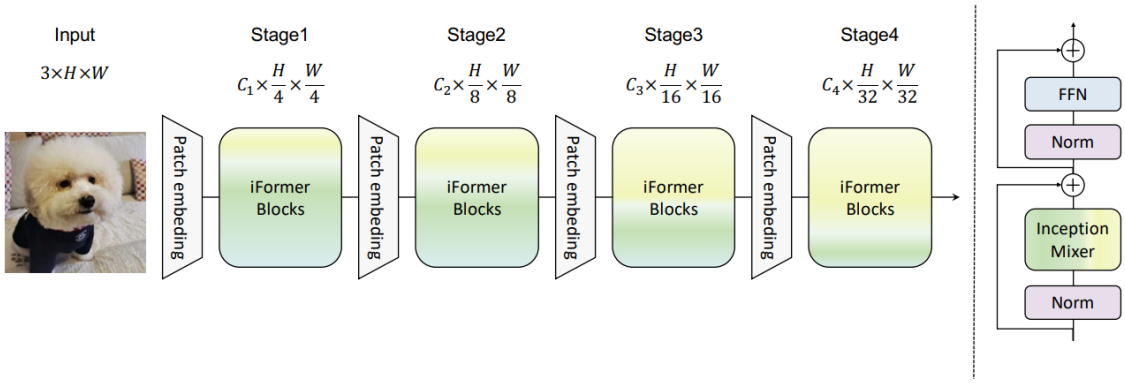

图 2: 整体结构

Stage	Layer	iFormer-S
1	Patch Embedding	3×3, stride 2, 48 3×3, stride 2, 96
	iFormer Block	$\begin{bmatrix} C_h/h = 2/3 \\ C_l/h = 1/3 \\ \text{pool stride 2} \end{bmatrix} \times 3$
2	Patch Embedding	2×2, stride 2, 192
	iFormer Block	$\begin{bmatrix} C_h/h = 1/2 \\ C_l/h = 1/2 \\ \text{pool stride 2} \end{bmatrix} \times 3$
3	Patch Embedding	2×2, stride 2, 320
	iFormer Block	$\begin{bmatrix} C_h/h = 3/10 \rightarrow 1/10 \\ C_l/h = 7/10 \rightarrow 9/10 \\ \text{pool stride 1} \end{bmatrix} \times 9$
4	Patch Embedding	2×2, stride 2, 384
	iFormer Block	$\begin{bmatrix} C_h/h = 1/12 \\ C_l/h = 11/12 \\ \text{pool stride 1} \end{bmatrix} \times 3$

图 3: 四个阶段

3.2 特征提取模块

在一般的视觉框架中，Low-level Layer 在捕获高频细节方面发挥更多作用，而 High-level Layer 在建模低频全局信息方面发挥更多作用，即 ResNet 的分层表示。与人类一样，通过捕获高频分量中的细节，较低层可以捕获视觉基本特征，同时也逐渐收集局部信息以实现对输入的全局理解。受到启发，设计了一种 Frequency ramp structure，该结构将更多通道维度分配给 low-frequency mixer，更少的通道维度分配给 high-frequency mixer。如图 4所示。

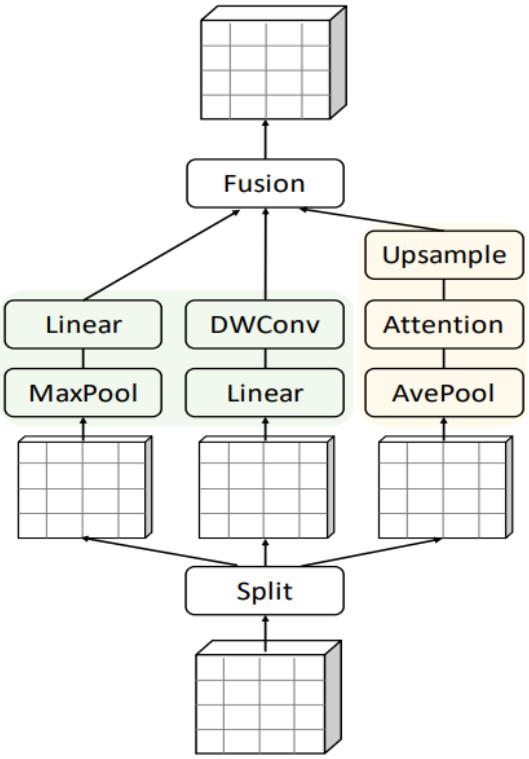

图 4: 具体改动模块

1、High-frequency mixer

考虑到最大滤波器的敏锐敏感性和卷积运算的细节感知，使用卷积神经网络来学习高频分量。首先沿通道划分后的高频 Xh 均分为 h1 和 h2。两者分别经过一个最大池化和一个线性层，被馈送到一个线性和一个深度卷积层，如下所示：

$$Yh1 = FC(MaxPool(Xh1))$$

$$Yh2 = DwConv(FC(Xh2))$$

2、Low-frequency mixer

iFormer 使用 MSA 在 low-frequency mixer 的所有 Token 之间传递信息。尽管注意力学习全局表示的能力很强，但特征图的大分辨率会在较低层带来巨大的计算成本。因此，简单地利用平均池化层在注意力操作之前减少空间尺度，并在注意力操作之后使用上采样层来恢复原始空间维度。这种设计大

大减少了计算开销，并使注意力操作集中在嵌入全局信息上。这个分支处理定义如下

$$Yl = Upsample(MSA(AvePooling(Xl)))$$

3.3 损失函数定义

论文中并未提及介绍其具体使用的损失函数，具体源代码分析来看使用的是交叉熵损失函数。

4 复现细节

4.1 与已有开源代码对比

除开 TokenMixer 部分，本篇论文的实验内容与该论文作者的上半年的一篇论文《PoolFormer: MetaFormer Is Actually What You Need for Vision》[6]工作实验内容基本一致。因此除开 TokenMixer 部分，其余皆直接使用作者上半年的代码 (https://github.com/sail-sg/poolformer)。此外本次复现 attention 模块的 token 二维压缩一维部分 patch embedding，使用的是原始 Vision transformer（Vit）结合论文附录的模型结构稍加调整实现,Vit 如图 5所示：

```python
class PatchEmbed(nn.Module):
    """ 2D Image to Patch Embedding
    """
    def __init__(self, img_size=224, patch_size=16, in_chans=3, embed_dim=768, norm_layer=None, flatten=
        super().__init__()
        # img_size = (img_size, img_size)
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        self.img_size = img_size
        self.patch_size = patch_size
        self.grid_size = (img_size[0] // patch_size[0], img_size[1] // patch_size[1])
        self.num_patches = self.grid_size[0] * self.grid_size[1]
        self.flatten = flatten
        # 输入通道，输出通道，卷积核大小，步长
        # C*H*W->embed_dim*grid_size*grid_size
        self.proj = nn.Conv2d(in_chans, embed_dim, kernel_size=patch_size, stride=patch_size)
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        B, C, H, W = x.shape
        assert H == self.img_size[0] and W == self.img_size[1], \
            f"Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{self.img_size[1]})."
        x = self.proj(x)
        if self.flatten:
            x = x.flatten(2).transpose(1, 2)  # BCHW -> BNC
        x = self.norm(x)
        return x
```

图 5: PatchEmbed

另外,参考本复现论文的源码,统一将 poolformer 代码中的 Position Embedding 独立出来,poolformer 中 Position Embedding 部分如图 6所示：

```python
class AddPositionEmb(nn.Module):
    """Module to add position embedding to input features
    """

    def __init__(
            self, dim=384, spatial_shape=[14, 14],
    ):
        super().__init__()
        if isinstance(spatial_shape, int):
            spatial_shape = [spatial_shape]
        assert isinstance(spatial_shape, Sequence), \
            f'"spatial_shape" must by a sequence or int, ' \
            f'get {type(spatial_shape)} instead.'
        if len(spatial_shape) == 1:
            embed_shape = list(spatial_shape) + [dim]
        else:
            embed_shape = [dim] + list(spatial_shape)
        self.pos_embed = nn.Parameter(torch.zeros(1, *embed_shape))

    def forward(self, x):
        return x + self.pos_embed
```

图 6: PositionEmbed

4.2 实验环境搭建

4.3 界面分析与使用说明

python>=3.6

torch>=1.7.0;

torchvision>=0.8.0;

pyyaml;

5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。本次实验训练模型所使用数据、超参数基本与论文提及一致。但是受限于硬件的不稳定性等情况，不同于论文中训练集连续训练 310 个 epoch，复现中的训练中断了几次。中断了再重新加载保存的中间训练结果进行训练，分别于第 45、58、167、198、226 个 epoch 因硬件重启而被中断。最终在 imagenet-1 上斩获 81.928 的准确率，相较于原论文中 83.362 有明显的 1.4 准确率下降。其最终的训练结果如下：

```
Test: [   0/391]  Time: 3.640s (3.640s,   35.17/s)  Loss:  0.4497 (0.4497)  Acc@1:  96.094 ( 96.094)  Acc@5:  99.219 ( 99.219)
Test: [  10/391]  Time: 0.180s (0.890s,  143.75/s)  Loss:  1.1219 (0.5749)  Acc@1:  77.344 ( 92.188)  Acc@5:  98.438 ( 98.651)
Test: [  20/391]  Time: 1.501s (0.845s,  151.52/s)  Loss:  0.6565 (0.7107)  Acc@1:  90.625 ( 87.798)  Acc@5:  97.656 ( 98.177)
Test: [  30/391]  Time: 0.180s (0.758s,  168.82/s)  Loss:  1.0592 (0.8082)  Acc@1:  83.594 ( 85.232)  Acc@5:  92.969 ( 97.379)
Test: [  40/391]  Time: 1.346s (0.759s,  168.62/s)  Loss:  0.6914 (0.7439)  Acc@1:  92.188 ( 87.233)  Acc@5:  95.312 ( 97.637)
Test: [  50/391]  Time: 0.182s (0.714s,  179.34/s)  Loss:  0.4416 (0.7427)  Acc@1:  96.094 ( 87.439)  Acc@5: 100.000 ( 97.518)
Test: [  60/391]  Time: 0.609s (0.711s,  180.13/s)  Loss:  0.9948 (0.7133)  Acc@1:  81.250 ( 88.256)  Acc@5:  95.312 ( 97.695)
Test: [  70/391]  Time: 0.184s (0.691s,  185.12/s)  Loss:  0.9524 (0.7425)  Acc@1:  82.031 ( 87.478)  Acc@5:  98.438 ( 97.623)
Test: [  80/391]  Time: 0.181s (0.707s,  180.98/s)  Loss:  0.8261 (0.7607)  Acc@1:  86.719 ( 87.105)  Acc@5:  96.875 ( 97.531)
Test: [  90/391]  Time: 0.181s (0.688s,  186.08/s)  Loss:  1.0115 (0.7671)  Acc@1:  74.219 ( 87.054)  Acc@5:  96.094 ( 97.476)
Test: [ 100/391]  Time: 0.181s (0.687s,  186.43/s)  Loss:  0.7253 (0.7747)  Acc@1:  89.844 ( 86.742)  Acc@5:  97.656 ( 97.532)
Test: [ 110/391]  Time: 0.554s (0.684s,  187.15/s)  Loss:  1.2474 (0.7868)  Acc@1:  60.938 ( 86.226)  Acc@5:  96.875 ( 97.572)
Test: [ 120/391]  Time: 0.182s (0.680s,  188.31/s)  Loss:  0.6408 (0.7846)  Acc@1:  91.406 ( 86.351)  Acc@5:  99.219 ( 97.605)
Test: [ 130/391]  Time: 0.344s (0.680s,  188.28/s)  Loss:  0.4757 (0.7750)  Acc@1:  96.094 ( 86.611)  Acc@5:  99.219 ( 97.620)
Test: [ 140/391]  Time: 0.181s (0.683s,  187.48/s)  Loss:  1.1826 (0.7751)  Acc@1:  75.781 ( 86.480)  Acc@5:  99.219 ( 97.695)
Test: [ 150/391]  Time: 0.181s (0.679s,  188.57/s)  Loss:  0.9061 (0.7826)  Acc@1:  78.125 ( 86.357)  Acc@5:  98.438 ( 97.630)
Test: [ 160/391]  Time: 0.181s (0.685s,  186.97/s)  Loss:  0.9017 (0.7859)  Acc@1:  84.375 ( 86.301)  Acc@5:  95.312 ( 97.559)
Test: [ 170/391]  Time: 0.181s (0.679s,  188.56/s)  Loss:  1.6643 (0.8056)  Acc@1:  64.062 ( 85.791)  Acc@5:  88.281 ( 97.341)
Test: [ 180/391]  Time: 0.181s (0.683s,  187.36/s)  Loss:  1.4846 (0.8208)  Acc@1:  63.281 ( 85.398)  Acc@5:  96.094 ( 97.181)
Test: [ 190/391]  Time: 0.182s (0.682s,  187.77/s)  Loss:  1.4753 (0.8429)  Acc@1:  67.188 ( 84.813)  Acc@5:  92.969 ( 96.981)
Test: [ 200/391]  Time: 0.182s (0.683s,  187.40/s)  Loss:  0.7818 (0.8617)  Acc@1:  85.156 ( 84.375)  Acc@5:  97.656 ( 96.770)
Test: [ 210/391]  Time: 0.181s (0.683s,  187.37/s)  Loss:  1.0778 (0.8745)  Acc@1:  79.688 ( 84.042)  Acc@5:  95.312 ( 96.653)
Test: [ 220/391]  Time: 0.182s (0.684s,  187.25/s)  Loss:  0.5972 (0.8761)  Acc@1:  89.062 ( 83.968)  Acc@5:  98.438 ( 96.606)
Test: [ 230/391]  Time: 0.181s (0.679s,  188.63/s)  Loss:  1.4413 (0.8818)  Acc@1:  74.219 ( 83.885)  Acc@5:  92.188 ( 96.550)
Test: [ 240/391]  Time: 0.184s (0.681s,  188.09/s)  Loss:  0.8133 (0.8825)  Acc@1:  88.281 ( 83.963)  Acc@5:  93.750 ( 96.480)
Test: [ 250/391]  Time: 0.182s (0.675s,  189.60/s)  Loss:  0.5980 (0.8988)  Acc@1:  92.188 ( 83.451)  Acc@5:  98.438 ( 96.330)
Test: [ 260/391]  Time: 0.182s (0.678s,  188.71/s)  Loss:  0.8691 (0.9097)  Acc@1:  82.031 ( 83.178)  Acc@5:  97.656 ( 96.187)
Test: [ 270/391]  Time: 0.182s (0.674s,  189.90/s)  Loss:  1.4991 (0.9162)  Acc@1:  63.281 ( 82.991)  Acc@5:  94.531 ( 96.131)
Test: [ 280/391]  Time: 0.181s (0.677s,  189.20/s)  Loss:  1.0386 (0.9175)  Acc@1:  78.906 ( 82.988)  Acc@5:  95.312 ( 96.102)
Test: [ 290/391]  Time: 0.182s (0.673s,  190.19/s)  Loss:  1.4000 (0.9229)  Acc@1:  55.469 ( 82.850)  Acc@5:  93.750 ( 96.032)
Test: [ 300/391]  Time: 0.182s (0.672s,  190.38/s)  Loss:  0.8466 (0.9291)  Acc@1:  87.500 ( 82.737)  Acc@5:  97.656 ( 95.954)
Test: [ 310/391]  Time: 0.182s (0.669s,  191.27/s)  Loss:  0.9395 (0.9325)  Acc@1:  83.594 ( 82.662)  Acc@5:  94.531 ( 95.885)
Test: [ 320/391]  Time: 0.181s (0.674s,  189.96/s)  Loss:  0.7047 (0.9392)  Acc@1:  89.062 ( 82.520)  Acc@5:  96.875 ( 95.814)
Test: [ 330/391]  Time: 0.181s (0.670s,  190.98/s)  Loss:  1.3161 (0.9490)  Acc@1:  68.750 ( 82.255)  Acc@5:  93.750 ( 95.728)
Test: [ 340/391]  Time: 0.181s (0.673s,  190.20/s)  Loss:  0.7354 (0.9512)  Acc@1:  85.156 ( 82.187)  Acc@5:  97.656 ( 95.707)
Test: [ 350/391]  Time: 0.182s (0.669s,  191.30/s)  Loss:  0.8627 (0.9524)  Acc@1:  82.812 ( 82.178)  Acc@5:  97.656 ( 95.709)
Test: [ 360/391]  Time: 0.182s (0.672s,  190.38/s)  Loss:  1.3537 (0.9587)  Acc@1:  73.438 ( 81.999)  Acc@5:  94.531 ( 95.667)
Test: [ 370/391]  Time: 0.182s (0.669s,  191.41/s)  Loss:  1.1776 (0.9569)  Acc@1:  67.188 ( 81.989)  Acc@5:  95.312 ( 95.692)
Test: [ 380/391]  Time: 0.181s (0.673s,  190.16/s)  Loss:  0.9086 (0.9593)  Acc@1:  82.812 ( 81.931)  Acc@5:  95.312 ( 95.673)
Test: [ 390/391]  Time: 0.254s (0.672s,  119.11/s)  Loss:  1.5534 (0.9576)  Acc@1:  60.000 ( 81.928)  Acc@5:  92.500 ( 95.700)
img_size224.000 * Acc@1 81.928 (18.072) Acc@5 95.700 (4.300)
```

图 7: 结果

准确率下降的原因一方面可以归为复现中复现参数的不够多，由原论文 19.87M 变动至 18.61M 带来的模型容量变化。另一方面也可归为训练被多次中断又重新加载断点训练带来的训练损耗。

6 总结与展望

本次论文复现的工作使我更加具体细节地认识到一篇顶刊论文工作量的庞大与严谨性。实验后续可能的提升尝试主要集中在加入频率注意力机制，使得高低频信息能够被更好地凸显并得以处理。

参考文献

[1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention Is All You Need[J]. arXiv, 2017.

[2] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale[J]. ICLR, 2021.

[3] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998.

[4] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition [J]. Computer Science, 2014.

[5] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[J]. IEEE, 2016.

[6] YU W, LUO M, ZHOU P, et al. Metaformer is actually what you need for vision[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 10819-10829.