浅谈CMT以及复现

> 论文链接: https://arxiv.org/abs/2107.06263
> 论文代码(个人实现版本): https://github.com/FlyEgle/CMT-pytorch

写在前面

本篇博客讲解CMT模型并给出从0-1复现的过程以及实验结果，由于论文的细节并没有给出来，所以最后的复现和paper的精度有一点差异，等作者release代码后，我会详细的校对我自己的code，找找原因。

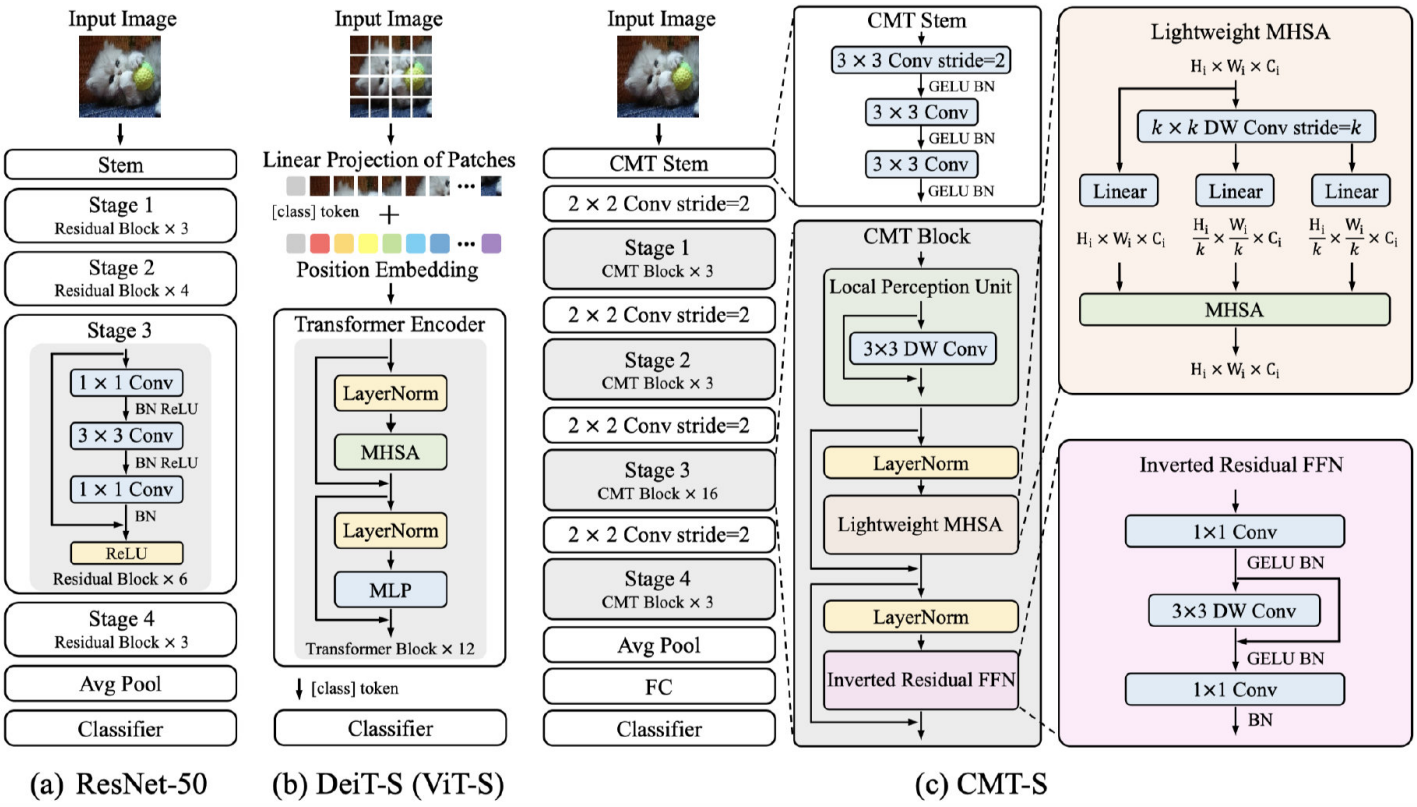
1. 出发点

- Transformers与现有的卷积神经网络（CNN）在性能和计算成本方面仍有差距。
- 希望提出的模型不仅可以超越典型的Transformers，而且可以超越高性能卷积模型。

2. 怎么做

1. 提出混合模型(串行)，通过利用Transformers来捕捉长距离的依赖关系，并利用CNN来获取局部特征。
2. 引入depth-wise卷积，获取局部特征的同时，减少计算量
3. 使用类似R50模型结构一样的stageblock，使得模型具有下采样增强感受野和迁移dense的能力。
4. 使用conv-stem来使得图像的分辨率缩放从VIT的1/16变为1/4，保留更多的patch信息。

3. 模型结构



- (a)表示的是标准的R50模型，具有4个stage，每个都会进行一次下采样。最后得到特征表达后，经过AvgPool进行分类
- (b)表示的是标准的VIT模型，先进行patch的划分，然后embeeding后进入Transformer的block，这里，由于Transformer是long range的，所以进入什么，输出就是什么，引入了一个非image的class token来做分类。
- (c)表示的是本文所提出的模型框架CMT，由CMT-stem, downsampling, cmt block所组成，整体结构则是类似于R50，所以可以很好的迁移到dense任务上去。

3.1. CMT Stem

使用convolution来作为transformer结构的stem，这个观点FB也有提出一篇paper，Early Convolutions Help Transformers See Better。

CMT&Conv stem共性

- 使用4层conv3x3+stride2 + conv1x1 stride 1 等价于VIT的patch embeeding，conv16x16 stride 16.
- 使用conv stem，可以使模型得到更好的收敛，同时，可以使用SGD优化器来训练模型，对于超参数的依赖没有原始的那么敏感。好处那是大大的多啊，仅仅是改了一个conv stem。

CMT&Conv stem异性

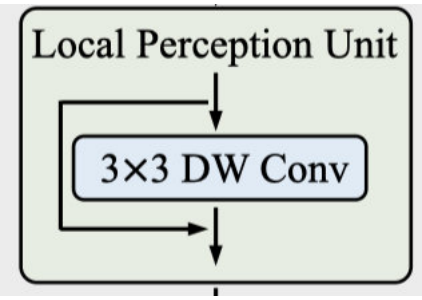
- 本文仅仅做了一次conv3x3 stride2，实际上只有一次下采样，相比conv stem，可以保留更多的patch的信息到下层。

从时间上来说，一个20210628(conv stem)， 一个是20210713(CMT stem)，存在借鉴的可能性还是比较小的，也说明了conv stem的确是work。

3.2. CMT Block

每一个stage都是由CMT block所堆叠而成的，CMT block由于是transformer结构，所以没有在stage里面去设计下采样。每个CMT block都是由 `Local Perception Unit, Lightweight MHSA, Inverted Residual FFN` 这三个模块所组成的，下面分别介绍：

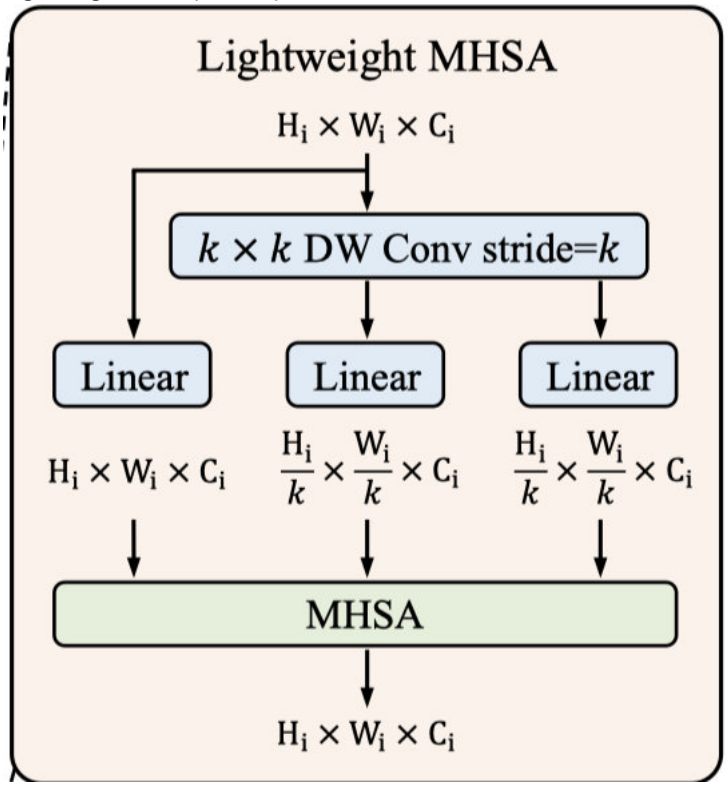
- **Local Perception Unit(LPU)**



本文的一个核心点是希望模型具有long-range的能力，同时还要具有local特征的能力，所以提出了LPU这个模块，很简单，一个3X3的DWconv，来做局部特征，同时减少点计算量，为了让Transformers的模块获取的longrange的信息不缺失，这里做了一个shortcut，公式描述为:

$$LPU(X) = DWConv(X) + X$$

- **Lightweight MHSA(LMHSA)**



MHSA这个不用多说了，多头注意力，L ightweight这个作用，PVT曾经有提出过，目的是为了降低复杂度，减少计算量。那本文是怎么做的呢，很简

单，假设我们的输入为$H_i \times W_i \times C_i$, 对其分别做一个scale，使用卷积核为$k \times k$，stride为$k$的Depth Wise卷积来做了一次下采样，得到的shape为$\frac{H_i}{k} \times \frac{W_i}{k} \times C_i$，那么对应的Q,K,V的shape分别为:

$$
\begin{aligned}
Q_{shape} &= (H_i \times W_i) \times C_i = N_i \times C_i \\
K_{shape} &= (\frac{H_i}{k_i} \times \frac{W_i}{k_i}) \times C_i = N_I^{'} \times C_i \\
V_{shape} &= (\frac{H_i}{k_i} \times \frac{W_i}{k_i}) \times C_i = N_I^{'} \times C_i
\end{aligned}
$$

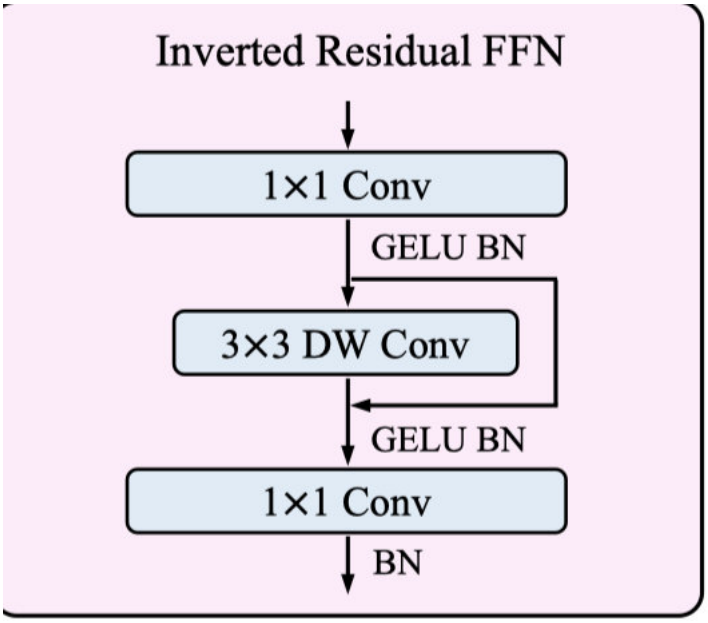
我们知道，在计算MHSA的时候要遵守两个计算原则:

1. Q, K的序列dim要一致。
2. K, V的token数量要一致。

所以，本文中的MHSA计算公式如下:

$$
LeightweightMHSA(Q, K, V) = Softmax(\frac{QK^{'T}}{\sqrt{d_k}} + B)V^{'}
$$

- **Inverted Resdiual FFN(IRFFN)**



FFN的这个模块，其实和mbv2的block基本上就是一样的了，不一样的地方在于，使用的是GELU，采用的也是DW+PW来减少标准卷积的计算量。很简单，就不多说了，公式如下:

$$
\begin{aligned}
IRFFN(X) &= Conv(F(Conv(X))) \\
F(X) &= DWConv(X) + X
\end{aligned}
$$

那么我们一个block里面的整体计算公式如下:

$$
\begin{aligned}
X_i^{'} &= LPU(X_{i-1}) \\
X_i^{''} &= LMHSA(LN(X_i^{'})) + X_i^{'} \\
X_i &= IRFFN(LN(X_i^{''})) + X_i^{''}
\end{aligned}
$$

3.3 patch aggregation

每个stage都是由上述的多个CMTblock所堆叠而成, 上面也提到了，这里由于是transformer的操作，不会设计到scale尺度的问题，但是模型需要构造下采样，来实现层次结构，所以downsampling的操作单独拎了出来，每个stage之前会做一次卷积核为2x2的，stride为2的卷积操作，以达到下采样的效果。

所以，整体的模型结构就一目了然了，假设输入为224x224x3,经过CMT-STEM和第一次下采样后，得到了一个56x56的featuremap，然后进入stage1，输出不变，经过下采样后，输入为28x28，进入stage2，输出后经过下采样，输入为14x14，进入stage3，输出后经过最后的下采样，输入为7x7，进入stage4，最后输出7x7的特征图，后面接avgpool和分类，达到分类的效果。

我们接下来看一下怎么复现这篇paper。

4. 论文复现

ps: 这里的**复现**指的是没有源码的情况下，实现网络，训练等，如果是结果复现，会标明为**复现精度**。

这里存在几个问题

- 文章的问题：我看到paper的时候，是第一个版本的arxiv，大概过了一周左右V2版本放出来了，这两个版本有个很大的diff。

Output Size	Layer Name	CMT-Ti	CMT-XS	CMT-S	CMT-B
112×112	Stem	$3 \times 3, 16$, stride 2 $[3 \times 3, 16] \times 2$	$3 \times 3, 16$, stride 2 $[3 \times 3, 16] \times 2$	$3 \times 3, 32$, stride 2 $[3 \times 3, 32] \times 2$	$3 \times 3, 38$, stride 2 $[3 \times 3, 38] \times 2$
56×56	Patch Aggr.	$2 \times 2, 46$, stride 2	$2 \times 2, 52$, stride 2	$2 \times 2, 64$, stride 2	$2 \times 2, 76$, stride 2
Stage 1	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 46 \\ H_1{=}1, k_1{=}8 \\ R_1{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 52 \\ H_1{=}1, k_1{=}8 \\ R_1{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ H_1{=}1, k_1{=}8 \\ R_1{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 76 \\ H_1{=}1, k_1{=}8 \\ R_1{=}4 \end{bmatrix} \times 4$
28×28	Patch Aggr.	$2 \times 2, 76$, stride 2	$2 \times 2, 96$, stride 2	$2 \times 2, 128$, stride 2	$2 \times 2, 160$, stride 2
Stage 2	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 92 \\ H_2{=}2, k_2{=}4 \\ R_2{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 104 \\ H_2{=}2, k_2{=}4 \\ R_2{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 128 \\ H_2{=}2, k_2{=}4 \\ R_2{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 152 \\ H_2{=}2, k_2{=}4 \\ R_2{=}4 \end{bmatrix} \times 4$
14×14	Patch Aggr.	$2 \times 2, 152$, stride 2	$2 \times 2, 192$, stride 2	$2 \times 2, 256$, stride 2	$2 \times 2, 320$, stride 2
Stage 3	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 184 \\ H_3{=}4, k_3{=}2 \\ R_3{=}3.6 \end{bmatrix} \times 10$	$\begin{bmatrix} 3 \times 3, 208 \\ H_3{=}4, k_3{=}2 \\ R_3{=}3.8 \end{bmatrix} \times 12$	$\begin{bmatrix} 3 \times 3, 256 \\ H_3{=}4, k_3{=}2 \\ R_3{=}4 \end{bmatrix} \times 16$	$\begin{bmatrix} 3 \times 3, 304 \\ H_3{=}4, k_3{=}2 \\ R_3{=}4 \end{bmatrix} \times 20$
7×7	Patch Aggr.	$2 \times 2, 304$, stride 2	$2 \times 2, 384$, stride 2	$2 \times 2, 512$, stride 2	$2 \times 2, 640$, stride 2
Stage 4	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 368 \\ H_4{=}8, k_4{=}1 \\ R_4{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 416 \\ H_4{=}8, k_4{=}1 \\ R_4{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 512 \\ H_4{=}8, k_4{=}1 \\ R_4{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 608 \\ H_4{=}8, k_4{=}1 \\ R_4{=}4 \end{bmatrix} \times 4$
1×1	FC	$1 \times 1, 1280$			
1×1	Classifier	$1 \times 1, 1000$			
# Params		9.49 M	15.24 M	25.14 M	45.72 M
# FLOPs		0.64 B	1.54 B	4.04 B	9.33 B

Output Size	Layer Name	CMT-Ti	CMT-XS	CMT-S	CMT-B
112×112	Stem	$3 \times 3, 16$, stride 2 $[3 \times 3, 16] \times 2$	$3 \times 3, 16$, stride 2 $[3 \times 3, 16] \times 2$	$3 \times 3, 32$, stride 2 $[3 \times 3, 32] \times 2$	$3 \times 3, 38$, stride 2 $[3 \times 3, 38] \times 2$
56×56	Patch Aggr.	$2 \times 2, 46$, stride 2	$2 \times 2, 52$, stride 2	$2 \times 2, 64$, stride 2	$2 \times 2, 76$, stride 2
Stage 1	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 46 \\ H_1{=}1, k_1{=}8 \\ R_1{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 52 \\ H_1{=}1, k_1{=}8 \\ R_1{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ H_1{=}1, k_1{=}8 \\ R_1{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 76 \\ H_1{=}1, k_1{=}8 \\ R_1{=}4 \end{bmatrix} \times 4$
28×28	Patch Aggr.	$2 \times 2, 92$, stride 2	$2 \times 2, 104$, stride 2	$2 \times 2, 128$, stride 2	$2 \times 2, 152$, stride 2
Stage 2	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 92 \\ H_2{=}2, k_2{=}4 \\ R_2{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 104 \\ H_2{=}2, k_2{=}4 \\ R_2{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 128 \\ H_2{=}2, k_2{=}4 \\ R_2{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 152 \\ H_2{=}2, k_2{=}4 \\ R_2{=}4 \end{bmatrix} \times 4$
14×14	Patch Aggr.	$2 \times 2, 184$, stride 2	$2 \times 2, 208$, stride 2	$2 \times 2, 256$, stride 2	$2 \times 2, 304$, stride 2
Stage 3	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 184 \\ H_3{=}4, k_3{=}2 \\ R_3{=}3.6 \end{bmatrix} \times 10$	$\begin{bmatrix} 3 \times 3, 208 \\ H_3{=}4, k_3{=}2 \\ R_3{=}3.8 \end{bmatrix} \times 12$	$\begin{bmatrix} 3 \times 3, 256 \\ H_3{=}4, k_3{=}2 \\ R_3{=}4 \end{bmatrix} \times 16$	$\begin{bmatrix} 3 \times 3, 304 \\ H_3{=}4, k_3{=}2 \\ R_3{=}4 \end{bmatrix} \times 20$
7×7	Patch Aggr.	$2 \times 2, 368$, stride 2	$2 \times 2, 416$, stride 2	$2 \times 2, 512$, stride 2	$2 \times 2, 608$, stride 2
Stage 4	LPU LMHSA IRFFN	$\begin{bmatrix} 3 \times 3, 368 \\ H_4{=}8, k_4{=}1 \\ R_4{=}3.6 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 416 \\ H_4{=}8, k_4{=}1 \\ R_4{=}3.8 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 512 \\ H_4{=}8, k_4{=}1 \\ R_4{=}4 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 608 \\ H_4{=}8, k_4{=}1 \\ R_4{=}4 \end{bmatrix} \times 4$
1×1	FC	$1 \times 1, 1280$			
1×1	Classifier	$1 \times 1, 1000$			
# Params		9.49 M	15.24 M	25.14 M	45.72 M
# FLOPs		0.64 B	1.54 B	4.04 B	9.33 B

网络结构可以说完全不同的情况下，FLOPs竟然一样的，当然可能是写错了，这里就不吐槽了。不过我一开始代码复现就是按下面来的，所以对于我也没影响多少，只是体验有点差罢了。

- 细节的问题：paper和很多的transformer一样，都是采用了Deit的训练策略，但是差别在于别的paper或多或少会给出来额外的tirck，比如最后FC的dp的ratio等，或者会改变一些，再不济会把代码直接release了，所以只好闷头尝试Trick。

4.1 复现难点

paper里面采用的Position Embeeding和Swin是类似的，都是Relation Position Bias，但是和Swin不相同的是，我们的Q,K,V尺度是不一样的。这里我考虑了两种实现方法，一种是直接bicubic插值，另一种则是切片，切片更加直观且embeeding我设置的可BP，所以，实现里面采用的是这种方法，代码如下：

```
def generate_relative_distance(number_size):
    """return relative distance, (number_size**2, number_size**2, 2)
    """
    indices = torch.tensor(np.array([[x, y] for x in range(number_size) for y in range(number_size)]))
    distances = indices[None, :, :] - indices[:, None, :]
    distances = distances + number_size - 1   # shift the zeros postion
    return distances
...
elf.position_embeeding = nn.Parameter(torch.randn(2 * self.features_size - 1, 2 * self.features_size - 1))

...
q_n, k_n = q.shape[1], k.shape[2]
attn = attn + self.position_embeeding[self.relative_indices[:, :, 0], self.relative_indices[:, :, 1]][:, :k_n]
```

4.2 复现trick历程(血与泪TT)

一方面想要看一下model是否是work的，一方面想要顺便验证一下DeiT的策略是否真的有效，所以从头开始做了很多的实验，简单整理如下：

- 数据:
  1. 训练数据: 20%的imagenet训练数据(快速实验)。
  2. 验证数据: 全量的imagenet验证数据。
- 环境:
  1. 8xV100 32G
  2. CUDA 10.2 + pytorch 1.7.1
- sgd优化器实验记录

model	augments	resolution	batchsize	epoch	optimizer	LR	strategy
CMT-TINY	crop+flip	184->160	512X8	120	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+mixup	184->160	512X8	120	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+cutmix	184->160	512X8	120	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+mixup	184->160	512X8	200	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+cutmix	184->160	512X8	300	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	200	SGD	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+ape(wrong->resolution)	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+rpe	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+ape(real->resolution)	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+pe_nd	1.6	cosine

model	augments	resolution	batchsize	epoch	optimizer	LR	strategy
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+qkv_bias	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+qkv_bias+rpe	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	SGD+qkv_bias+ape	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+no mixup+no_cutmix+labelsmoothing	184->160	512X8	300	SGD+qkv_bias+rpe	1.6	cosine
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	SGD+qkv_bias+rpe	1.6	cosine

结论: 可以看到在SGD优化器的情况下，使用1.6的LR，训练300个epoch，warmup5个epoch，是用cosine衰减学习率的策略，用randaug+colorjitter+mixup+cutmix+labelsmooth，设置weightdecay为0.1的配置下，使用QKV的bias以及相对位置偏差，可以达到比baseline高11%个点的结果，所有的实验都是用FP16跑的。

- adamw优化器实验记录

model	augments	resolution	batchsize	epoch	optimizer	
CMT-TINY	crop+flip	184->160	512X8	120	AdamW	4. 03
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	300	AdamW	4. 03
CMT-TINY	crop+flip+colorjitter+randaug	184->160	512X8	120	AdamW	4. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	4. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing + repsampler	184->160	512X8	300	adamw+qkv_bias+rpe	4. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	4. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	1. 04
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing + repsampler	184->160	512X8	300	adamw+qkv_bias+rpe	4. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	8. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	5. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	6. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	6. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing	184->160	512X8	300	adamw+qkv_bias+rpe	6. 03

model	augments	resolution	batchsize	epoch	optimizer	
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing+warmup20	184->160	512X8	300	adamw+qkv_bias+rpe	6. 03
CMT-TINY	crop+flip+colorjitter+randaug+mixup+cutmix+labelsmoothing+droppath	184->160	512X8	300	adamw+qkv_bias+rpe	6. 03

结论：使用AdamW的情况下，对学习率的缩放则是以512的bs为基础，所以对于4k的bs情况下，使用的是4e-3的LR，但是实验发现增大到6e-3的时候，还会带来一些提升，同时放大一点weightsdecay，也略微有所提升，最终使用AdamW的配置为，6e-3的LR，1e-1的weightdecay，和sgd一样的增强方法，然后加上了随机深度失活设置，最后比baseline高了16%个点，比SGD最好的结果要高0.8%个点。

4.3. imagenet上的结果

model-name	input_size	FLOPs	Params	acc@one_crop(ours)	acc(papers)	weights
CMT-T	160x160	516M	11.3M	75.124%	79.2%	weights

最后用全量跑，使用SGD会报nan的问题，我定位了一下发现，running_mean和running_std有nan出现，本以为是数据增强导致的0或者nan值出现，结果空跑几次数据发现没问题，只好把优化器改成了AdamW，结果上述所示，CMT-Tiny在160x160的情况下达到了75.124%的精度，相比MbV2,MbV3的确是一个不错的精度了，但是相比paper本身的精度还是差了将近4个点，很是离谱。

速度上，CMT虽然FLOPs低，但是实际的推理速度并不快，128的bs条件下，速度慢了R50将近10倍。

5. 实验结果

总体来说，CMT达到了更小的FLOPs同时有着不错的精度, imagenet上的结果如下：

Model	Top-1 Acc.	Top-5 Acc.	# Params	Resolution	# FLOPs	Ratio
CPVT-Ti-GAP [6]	74.9%	-	6M	224^2	1.3B	2.2×
DenseNet-169 [20]	76.2%	93.2%	14M	224^2	3.5B	5.8×
EfficientNet-B1 [50]	79.1%	94.4%	7.8M	240^2	0.7B	1.2×
CMT-Ti	**79.2%**	**94.6%**	9.5M	160^2	**0.6B**	**1**×
ResNet-50 [15]	76.2%	92.9%	25.6M	224^2	4.1B	2.7×
CoaT-Lite Mini [59]	78.9%	-	11M	224^2	2.0B	1.3×
DeiT-S [51]	79.8%	-	22M	224^2	4.6B	3.1×
EfficientNet-B3 [50]	81.6%	95.7%	12M	300^2	1.8B	1.2×
CMT-XS	**81.8%**	**95.8%**	15.2M	192^2	**1.5B**	**1**×
ResNeXt-101-64x4d [58]	80.9%	95.6%	84M	224^2	32B	8×
T2T-ViT-19 [62]	81.2%	-	39.0	224^2	8.0B	2×
PVT-M [54]	81.2%	-	44.2M	224^2	6.7B	1.7×
Swin-T [32]	81.3%	-	29M	224^2	4.5B	1.1×
CPVT-S-GAP [6]	81.5%	-	23M	224^2	4.6B	1.2×
RegNetY-8GF [40]	81.7%	-	39.2M	224^2	8.0B	2×
CeiT-S [61]	82.0%	95.9%	24.2M	224^2	4.5B	1.1×
CvT-13-NAS [57]	82.2%	-	18M	224^2	4.1B	1×
EfficientNet-B4 [50]	82.9%	96.4%	19M	380^2	4.2B	1×
Twins-SVT-B [5]	83.1%	-	56.0M	224^2	8.3B	2.1×
CMT-S	**83.5%**	**96.6%**	25.1M	224^2	**4.0B**	**1**×
ViT-B/16$_{\uparrow 384}$ [10]	77.9%	-	55.5M	384^2	77.9B	8.4×
T2T-ViT-24 [62]	82.2%	-	63.9M	224^2	12.6B	1.4×
CPVT-B [6]	82.3%	-	88M	224^2	17.6B	1.9×
TNT-B [13]	82.8%	96.3%	65.6M	224^2	14.1B	1.5×
DeiT-B$_{\uparrow 384}$ [51]	83.1%	-	85.8M	384^2	55.6B	6.0×
CvT-21$_{\uparrow 384}$ [57]	83.3%	-	31.5M	384^2	24.9B	2.7×
Swin-B [32]	83.3%	-	88M	224^2	15.4B	1.7×
Twins-SVT-L [5]	83.3%	-	99.2M	224^2	14.8B	1.6×
CeiT-S$_{\uparrow 384}$ [61]	83.3%	96.5%	24.2M	384^2	12.9B	1.4×
BoTNet-S1-128 [45]	83.5%	96.5%	75.1M	256^2	19.3B	2.1×
EfficientNet-B6 [50]	84.0%	96.8%	43M	528^2	19.2B	2.0×
CMT-B	**84.5%**	**96.9%**	45.7M	256^2	**9.3B**	**1**×
EfficientNet-B7 [50]	84.3%	97.0%	66M	600^2	37B	1.9×
CMT-L	**84.8%**	**97.1%**	74.7M	288^2	**19.5B**	**1**×

coco2017上也有这不错的精度

Table 7: **Object detection results on COCO val2017.** All models use RetinaNet as basic framework and are trained in "1x" schedule. FLOPs are calculated on 1280×800 input. † means the results are from [5].

Backbone	# Params	# FLOPs	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ConT-M [60]	217B	27.0M	37.9	58.1	40.2	23.0	40.6	50.4
ResNet-101 [15]	315B	56.7M	38.5	57.6	41.0	21.7	42.8	50.4
RelationNet++ [4]	266B	39.0M	39.4	58.2	42.5	-	-	-
ResNeXt-101-32x4d [58]	319B	56.4M	39.9	59.6	42.7	22.3	44.2	52.5
PVT-S [54]	226B	34.2M	40.4	61.3	43.0	25.0	42.9	55.7
Swin-T† [32]	245B	38.5M	41.5	62.1	44.2	25.1	44.9	55.5
Twins-SVT-S [5]	209B	34.3M	42.3	63.4	45.2	26.0	45.5	56.5
Twins-PCPVT-S [5]	226B	34.4M	43.0	64.1	46.0	27.5	46.3	57.3
CMT-S (ours)	231B	44.3M	**44.3**	**65.5**	**47.5**	**27.1**	**48.3**	**59.1**

6. 结论

本文提出了一种名为CMT的新型混合架构，用于视觉识别和其他下游视觉任务，以解决在计算机视觉领域以粗暴的方式利用Transformers的限制。所提出的CMT同时利用CNN和Transformers的优势来捕捉局部和全局信息，促进网络的表示能力。在ImageNet和其他下游视觉任务上进行的大量实验证明了所提出的CMT架构的有效性和优越性。

代码复现repo: https://github.com/FlyEgle/CMT-pytorch