Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

汤铭杰

摘要

SwinTransformer 框架区别于其他的基于 transformer 架构的模型引入了视觉信号中几种性质：层次性（hierarchy）、局部性（locality）、平移不变性（translation invariance)。图像分类任务是计算机视觉的三大任务之一，原论文把 Swin TransformerT 应用于 ImageNet1k 的分类任务上，取得了 81.3% 的准确率。本文复现的 Swin TransformerT，在 attention_mask 模块上加入了 baise，结果达到了 81% 的准确率。

关键词：关键词 1:SwinTransformer; 关键词 2:Transformer; 关键词 3: 计算机视觉;

1 引言

近年来，Transformer 在 NLP 中的巨大成功激发了研究它用于其他任务的研究。2020 年，将纯粹的 Transformer 体系结构应用视觉任务的 Vision Transformer（VIT）[1]引起了 AI 社区的广泛关注。2021 年微软研究人员发表了 Swin Transformer（Liu，2021）[2]，让 Transformer 与视觉信号的特点做结合，更好的建模视觉信号。相对于 ViT，SwinT 引入了视觉信号中几种性质：层次性（hierarchy）、局部性（locality）、平移不变性（translation invariance）。由于目前课题组的主要研究的方向是数字图像处理，而 swintranformer 一出现就在图像分类、目标检测、语义分割等多种机器视觉任务中达到了 SOTA 水平。所以想把 siwntranformer 结合到目前所做的方向，或者是想应用与目前的课题的 backbone，作为特征提取层。最后想通过学习 swintranformer 的 shifted windouws 的思想和对应的 msa-self attention，结合一种新的位置编码的思想，探索能否应用到课题组特定的下游任务中。

2 相关工作

2.1 注意力机制

目前大多数注意力模型附着在 Encoder-Decoder[3]框架下。其中 Encoder 用于将中文语句进行编码，这些编码后续将提供给 Decoder 进行使用；Decoder 将根据 Encoder 的数据进行解码。首先将前一个时刻的输出状态和 Encoder 的输出进行 Attention 计算，得到一个当前时刻的 context，用公式表示为:

$$[a_1, a_2, a_3, a_4] = softmax([s(q_2, h_1), s(q_2, h_2), s(q_2, h_3), s(q_2, h_4)])context = \sum_{n=1}^{4} a_i \cdot h_i$$

这里的 s[q,hj] 表示注意力打分函数，它是个标量，其大小描述了当前时刻在这些 Encoder 的结果上的关注程度。然后用 softmax 对这个结果进行归一化，最后使用加权求和获得当前时刻的上下文向量 context。这个 context 可以解释为：在此基础上下一个时刻应该更加关注源中文语句的那些内容。基与此基础，诞生了一些变体，如硬性注意力机制[4]，键值对注意力机制[5]，多头注意力机制[6]。

2.2 基于 transformer 的视觉模型

VIT[2](Vision Transformer) 是第一种将 transformer 运用到计算机视觉的网络架构，也是第一次将注意力机制也第一次[6]运用到了图片识别上面。虽然 ViT 取得了很好的指标，证明了 Transformer 可以作为统一 NLP 和 CV 领域比较通用的框架，但 VIT 并不完美，还是包含了几个缺点。ViT 工作忽略了视觉信号与文本信号之间的差异。这里面最严重的一个就是 VIT 和高分辨率图像的兼容性。VIT 并没有使用到图像的不同尺度的信息，以及考虑到视觉数据中的平移不变性。从而导致 Vit 的模型仅仅使用在图像的分类上面，而无法使用到更细粒度的问题，如图像分割和语义分割上。

3 本文方法

3.1 本文方法概述

整个模型采取层次化的设计，一共包含 4 个 Stage，每个 stage 都会缩小输入特征图的分辨率，像 CNN 一样逐层扩大感受野。在输入开始的时候，做了一个 PatchEmbedding，将图片切成一个个图块，并嵌入到 Embedding。在每个 Stage 里，由 Patch Merging 和多个 Block 组成。其中 PatchMerging 模块主要在每个 Stage 一开始降低图片分辨率。而 Block 主要是 LayerNorm，MLP，Window Attention 和 Shifted Window Attention 组成。引用原论文的一张图来介绍 swinTransform[2]的架构图 1所示：

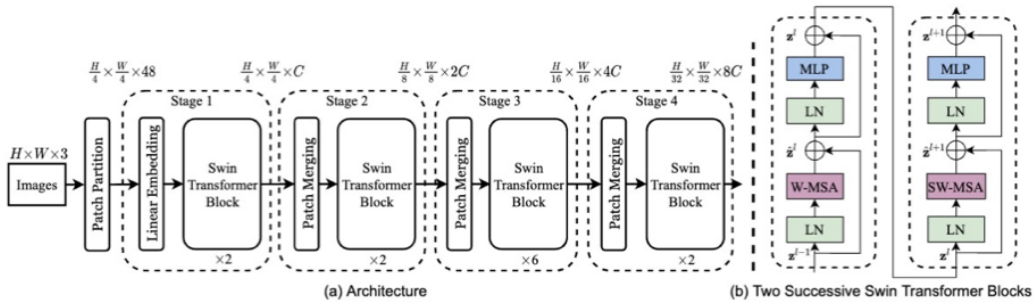

图 1: Swin Transform 的架构

3.2 特征提取模块

3.2.1 Patch Embedding

在输入进 Block 前，我们需要将图片切成一个个 patch，然后嵌入向量。具体做法是对原始图片裁成一个个 windowSize * windowSize 的窗口大小，然后进行嵌入。这里可以通过二维卷积层，将 stride，kernelsize 设置为 windowSize 大小。设定输出通道来确定嵌入向量的大小。最后将 H,W 维度展开，并移动到第一维度。2所示：

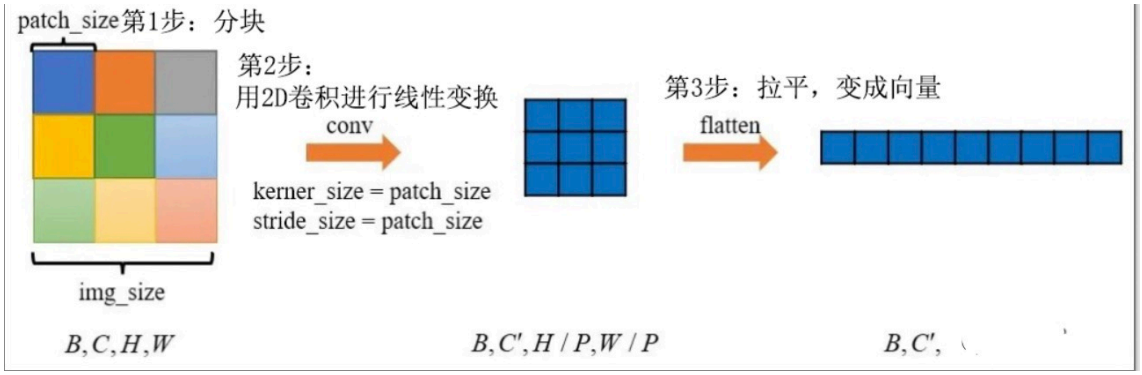

图 2: Patch Embedding

3.2.2 Patch Merging

该模块的作用是在每个 Stage 开始前做降采样，用于缩小分辨率，调整通道数进而形成层次化的设计，同时也能节省一定运算量。每次降采样是两倍，因此在行方向和列方向上，间隔 2 选取元素。然后拼接在一起作为一整个张量，最后展开。此时通道维度会变成原先的 4 倍（因为 H,W 各缩小 2 倍），此时再通过一个全连接层再调整通道维度为原来的两倍。实现：分别把图像的 w 和 h 维度切分一半，得到四个 patch，然后 torch.cat 方法，然后放入到全连接层。3所示：

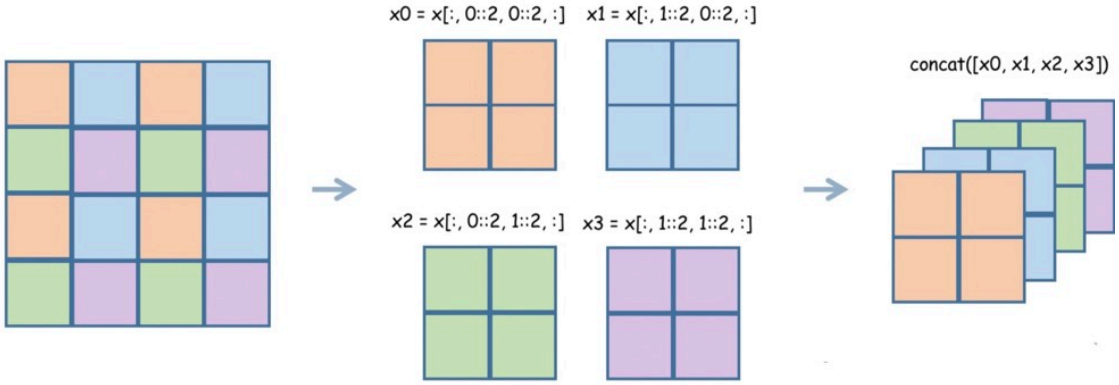

图 3: Patch Merging

3.2.3 Window Partition/Reverse

window partition 函数是用于对张量划分窗口，指定窗口大小。将原本的张量从 N H W C, 划分成 numWindows*B, windowSize, windowSize, C，其中 numWindows = H*W / windowSize，即窗口的个数。而 window reverse 函数则是对应的逆过程。这两个函数会在后面的 Window Attention 用到。

3.2.4 SW-MSA

移位窗口是指窗口划分往右下移动了半个窗口，如原论文图 4所示，既引入了跨窗口的信息流通，又保留了非重叠窗口的计算效率。在代码中，作者是用循环移位和 mask 结合来实现的，保证了计算的窗口数不增加，而且计算等价，通过一次前向过程就能完整所有的计算。

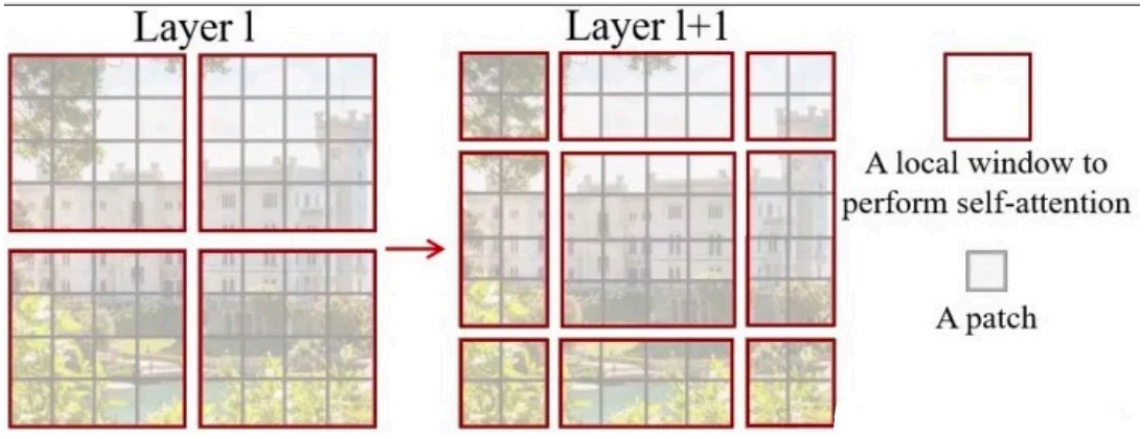

图 4: SW-MSA

3.3 损失函数定义

本实验使用交叉熵损失函数来描述。交叉熵（CrossEntropy）是 Shannon 信息论中一个重要概念，主要用于度量两个概率分布间的差异性信息。其公式是

$$L = \frac{1}{N}\sum_{i} L_i = -\frac{1}{N}\sum_{i}\sum_{c=1}^{M} y_i c \log(p_i c)$$

其中 M——类别的数目，y 符号函数 (0 或者 1)，如果样本 i 的真实类别等于 c 取 1，否则取 0，p 是观测样本 i 属于类别 c 的预测概率，此外，由于交叉熵涉及到计算每个类别的概率，所以最后加了个 softmax 函数。

3.4 优化器

本实验使用了 AdamW 来进行优化，AdamW 也就是 Adam + weight decate。如 AdamW 的原论文[7]图 5所示, 算法的描述：

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda\boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

图 5: AdamW

4 复现细节

4.1 与已有开源代码对比

原论文的开源代码写的太过复杂，model 层是自己模仿模型的框架写的。相比于开源代码，本文实现的代码更加简洁，但缺点是难扩展。transform 层的实现参考了源代码的实现，直接使用三层线性层来实现 qkv：qkv = self.qkv(x).reshape(B, N, 3, self.numHeads, C // self.numHheads).permute(2, 0, 3, 1, 4)。且在每层增加了的 baise 偏置，并且在优化器来回切换使用了 SGD 和 Adam 优化器。

对于 utils.py 是自己写的的代码来处理数据集的切割，使得数据集 train 被划分成 80% 用于训练，20% 用于测试, 具体的代码在 utils.py 的。并且创建一个 class——indices.json 文件, 该文件记录了预测的序号和类别的对应关系，如”1”: ”n01443537”,”2”: ”n01484850”... 等

对于 train.py 的文件是自己实现的，但和源码相比简单了一点，并没有实现动态管理 lr 的变化和动态调整超参数的过程，而是采用了简单实现加手动调参的方法，当然结果也没原论文的结果好。对于 predict 是普通的预测 1000 类的代码，是自己实现。

综上，本文实现的代码相比于官网分模块的实现，我的代码实现更加简洁，就只有 tanin.py 来训练模型，predict.py 来预测模型的输出，model.py 来创建 swinTransformer 模型，utils.py 是划分数据集，

my_dataset.py 是读取数据集。尽管扩展性，复用性和结果并不如源码，但本文实现的代码易阅读和上手。十分适合新手阅读。

4.2 实验环境搭建

采用的是 pytorch 框架版本是 11.7，python 版本是 3.9，使用到的包有:tqdm,matplotlib,PrettyTable,numpy, 在 RTX3090 上共训练了 7 天。数据集采用的是 ImageNet 1k。

4.3 整体的流程

第一，将一个 RGB 图像（H * W * 3）输入 Patch Partition 模块中进行切分。patch size 为 4 * 4，然后在 channel 维度上展平，因此每个 patch 的特征尺寸为 4 * 4 * 3 = 48。所以通过 Patch Partition 后图像 shape 由 [H, W, 3] 变成了 [H/4, W/4, 48]。第二，通过 LinearEmbeding 层对每个像素的 channel 数据做线性变换，由 48 变成 C，即图像 shape 再由 [H/4, W/4, 48] 变成了 [H/4, W/4, C]。第三，得到的 patch tokens 应用于 Swin Transformer blocks。Transformer blocks 保持 token 的大小为 H/4 * W/4，与 Linear Embedding 构成了“Stage 1”。为了产生层次表示，token 的数量随着网络的加深由 patch merging layers 减少。第四，patch merging layer 会将相邻像素划分为一个 patch。接着应用一个 linear layer 在深度方向 4C-dimensional 进行特征 concat 拼接。这将 token 的数量减少了 2 * 2 =4(2 倍下采样)，输出维度变成 2C。然后重复改过程，区别在于 patch merging layer 每次都会下采样二倍，channel 数会翻倍，最后接一个全连接层用于预测图像的分类。

5 实验结果分析

本文实现 1k 分类的准率率在测试集上达到 81%，比原论文的准确率低了 1%。第一是只使用了 ImageNET1K 的 train 的数据集，本文的 val 数据是从 tarin 数据集划分出来的，并不是 ImageNET1K 上的 val 的数据集第二是多用了 baise，结果却起到反作用 basie 的偏置在 attention_mask 的模块上。第三数据集太大了，加上调参技术不过关，很多时候要等一天才知道所调的超参数是否正确。

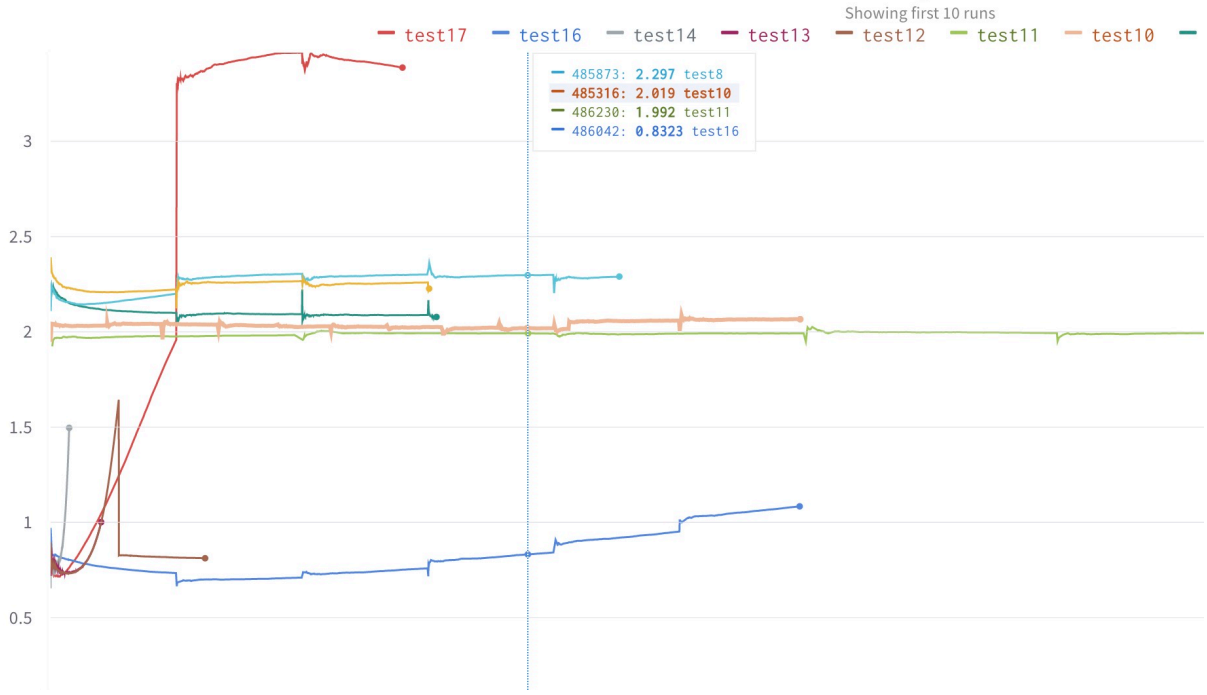

图 6: 训练误差

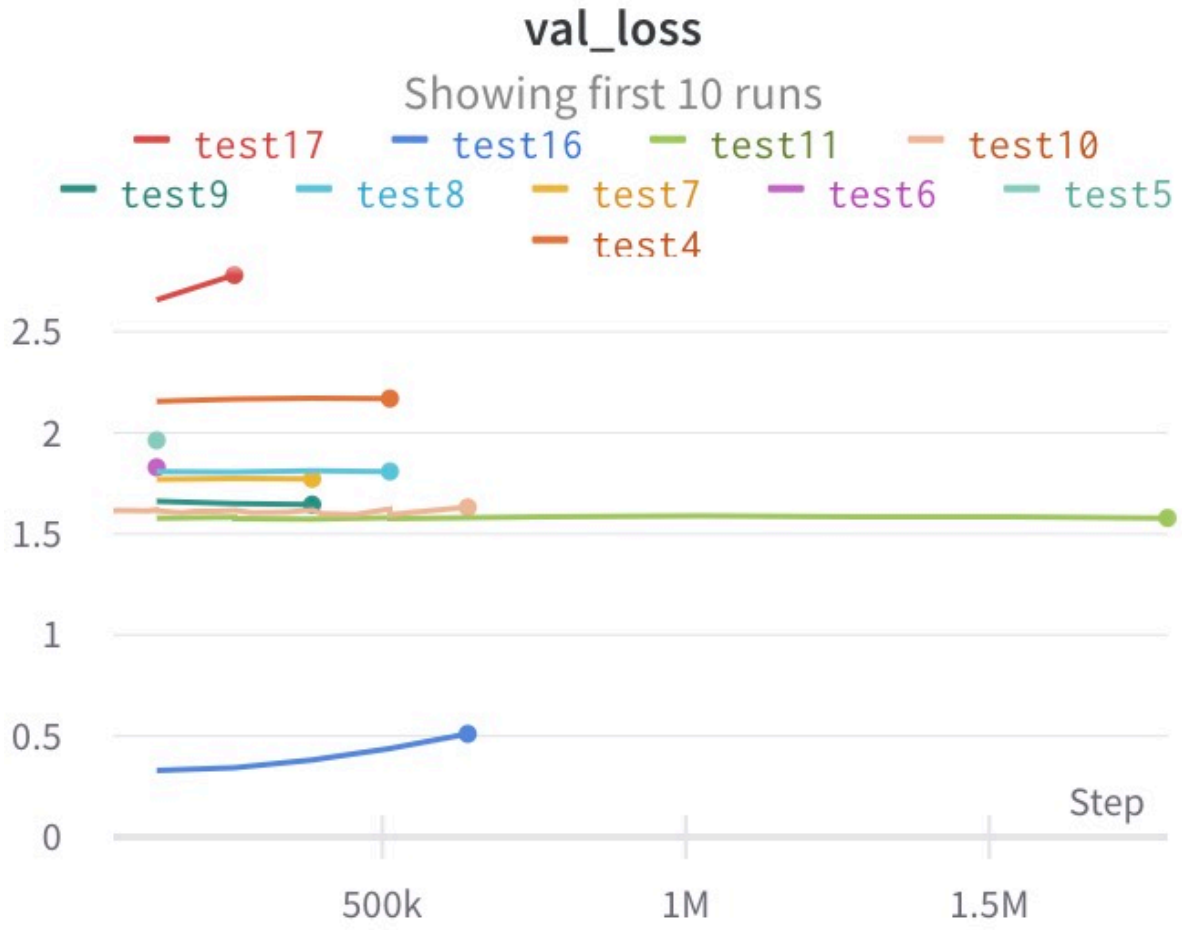

图 7: 测试误差

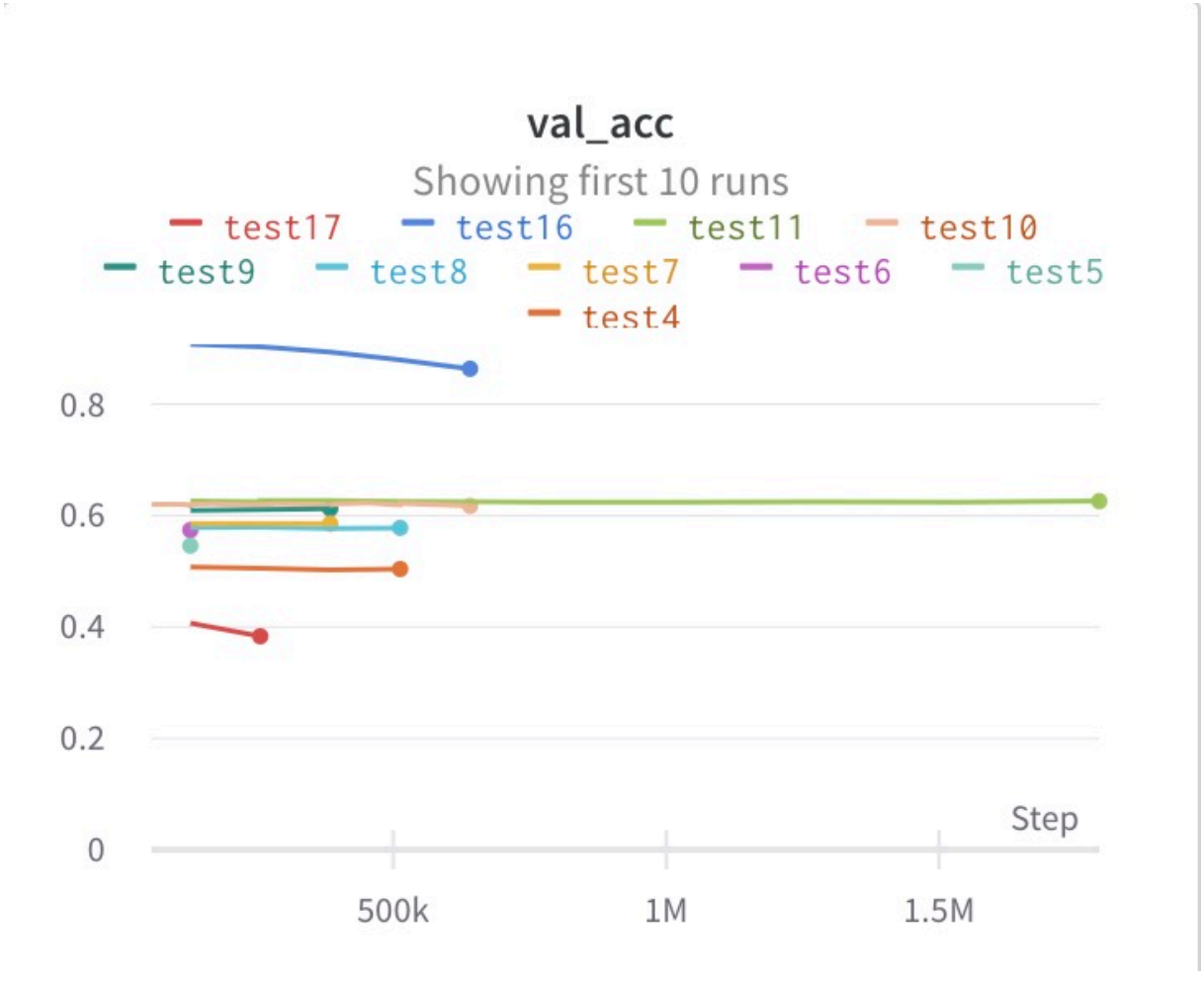

图 8: 训练准确率

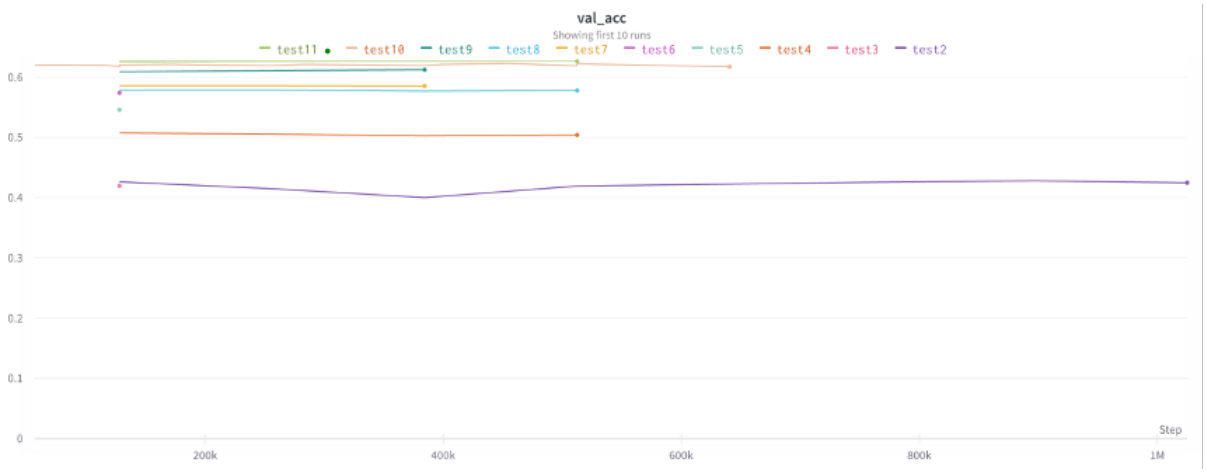

图 9: 测试的准确率

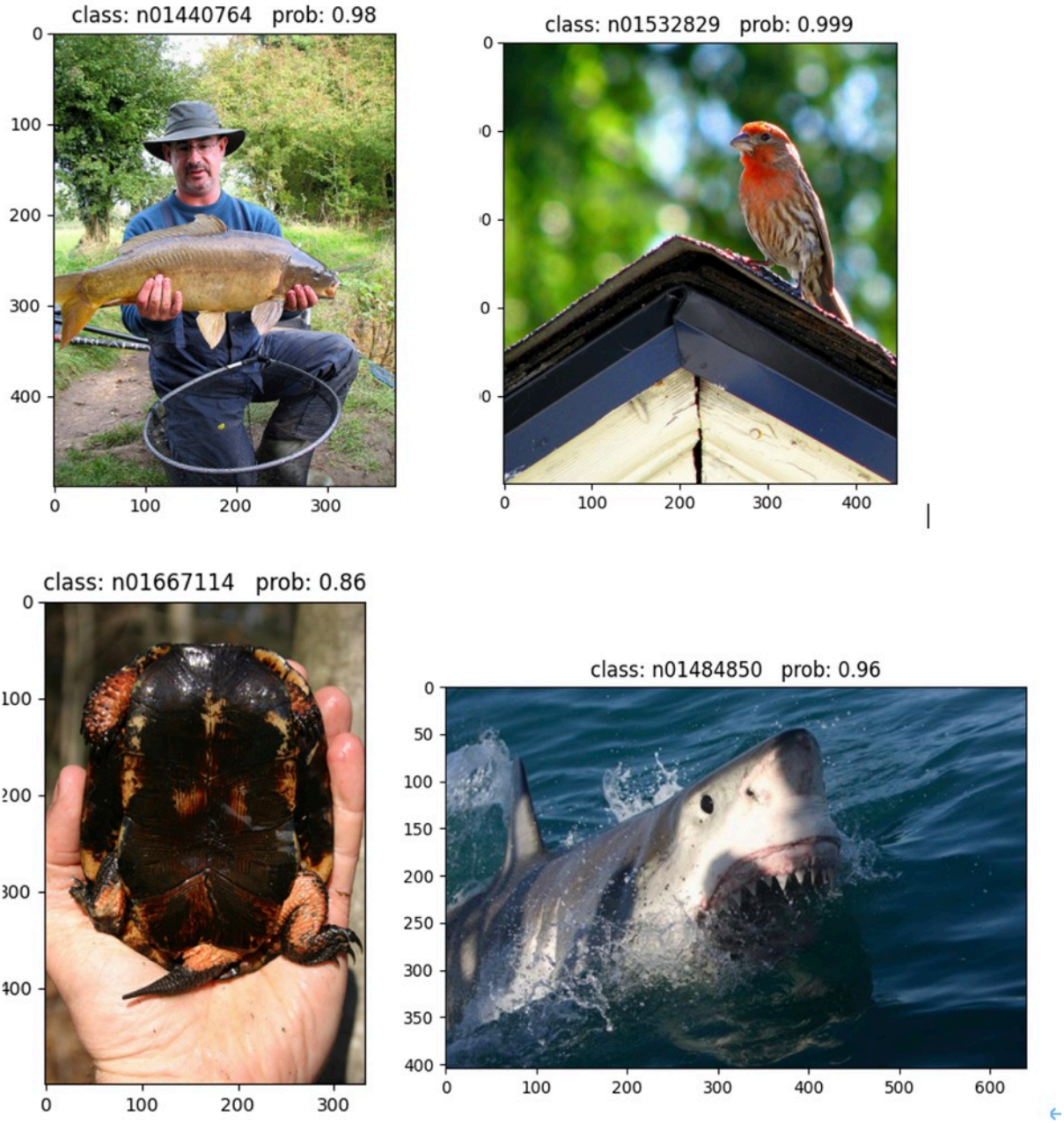

图 10: precict 的结果

6 总结与展望

SwinTransformer 像 CNN 一样具有层级式结构，而且它的计算复杂度是跟输入图像的大小呈线性增长的。这种层级式结构使得它在下游任务上的表现非常好，如 Swin Transformerr 在 COCO 和 ADE20K

上的效果都非常的好，远远超越了之前最好的方法。最大的创新就是使用了基于 Shifted Window 的自注意力，可以看到，相比于全局自注意力，它在有效地减少了的计算量的同时，还保持了很好的效果，因此对很多视觉的任务，尤其是对下游密集预测型的任务是非常有帮助的。本文实现的代码，后续可直接应用于自己的项目当中。通过这次的复现，使得我快速的上手深度学习的学习当中，对深度学习有个一个大概的了解，不在是一个懵懂少年，更加顺手的投入到后续的研究当中。

参考文献

[1] DOSOVITSKIY A, BEYER L, KOLESNIKOV A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale[J]. Learning, 2020.

[2] LIU Z, LIN Y, CAO Y, et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows.[J]. international conference on computer vision, 2021.

[3] CHO K, van MERRIËNBOER B, GULCEHRE C, et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation[J]. empirical methods in natural language processing, 2014.

[4] VINYALS O, TOSHEV A, BENGIO S, et al. Show and tell: A neural image caption generator[J]. computer vision and pattern recognition, 2015.

[5] CHAUDHARI S, POLATKAN G, RAMANATH R, et al. An Attentive Survey of Attention Models[J]. arXiv: Learning, 2019.

[6] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is All you Need[J]. neural information processing systems, 2017.

[7] LOSHCHILOV I, HUTTER F. Decoupled Weight Decay Regularization[J]. Learning, 2017.