《信号与系统》课程作业

学院：信息科学与技术学院

姓名：习钟

学号：20212108032

Vision Transformer 论文评述

序

从学习最新文章的角度来讲，AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE（下面都用ViT来代指这篇论文及模型），无疑是在这个时间点必须要看的文章之一，在图像处理的领域，尤其是图像分类，这篇文章引领了过去一年(2020-2021)中的潮流，这一年里的很多新研究都是ViT的变种，包括今年的Swin Transformer,所以ViT成了不能不看的论文之一。

Transformer用于图像处理的难处

Transformer在NLP已经基本成为标准了，但是在图像领域要用Transformer的话，关键点在于Transformer里最主要的是自注意力操作，也就是需要两两相互进行互动计算，算出Attention，然后用Attention的图去做加权平均，最后得到输出。处理文本序列的时候都是以一维的形式来呈现的，但图像作为一个二维的对象，如果用像素点当作输入的话，势必会导致展开的序列太长，序列太长在目前的硬件上还无法做到能进行计算，所以这个问题有过以下解决思路：

1. 用网络中间生成的特征图当作Transformer的直接输入。
2. 用Stand Alone Attention和Axial Attention，前者用局部小窗口来控制计算复杂度，后者把图拆分成了高和宽两个1维的序列进行计算，但这样的话，还是没用到硬件加速，只适合一些小模型。 所以在大规模的图像识别上，还是传统的ResNet网络有这更好的识别效果。

ViT提出的解决方法

第一步：将图像进行切割

我们假设现在有一张图片是：224*224

1. 如果直接展开的话：224*224= 50776，这个序列对于Transformer来讲太大了。
2. 如果用ViT的方式将图像进行切割成16*16张小图片，那么小图片的长宽就是：224/16=14，展开就是14 * 14=196，这个序列对于Transformer来讲就完全没问题。

第二步：切割完并使用sequence of linear embeddings将图片作为输入进Transformer即可，这样看来，就相当于把Transformer在NLP中使用的单词换成了小的图片序列。

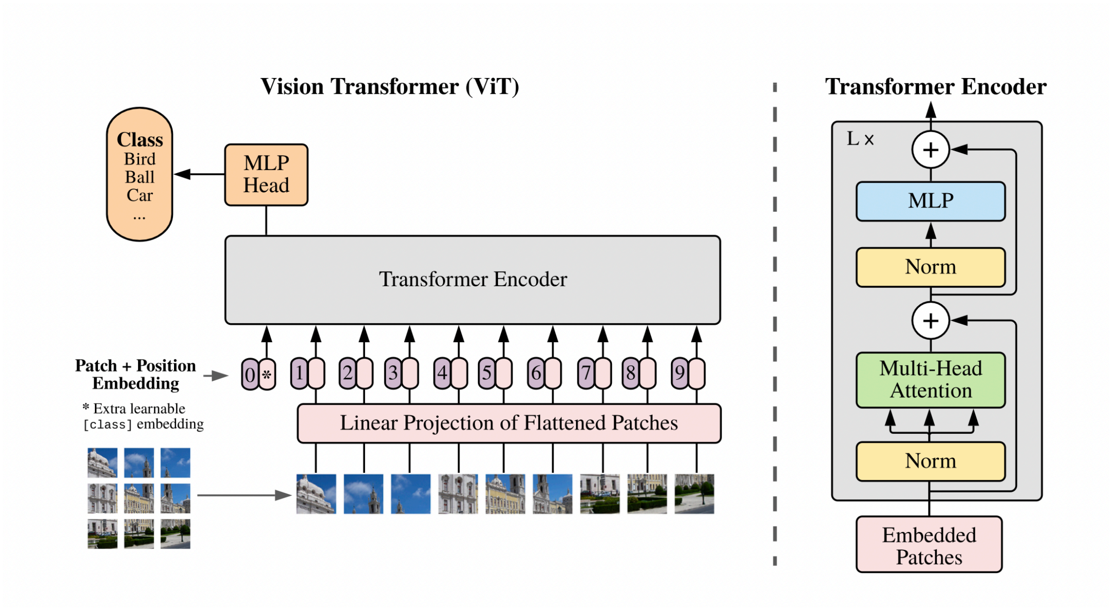
但是训练的时候，不同于NLP中常用的无监督训练，ViT在这里使用的是有监督的训练。

在这之前，也有文章(Cordonnier et al. (2020))有这样的处理思路，但是这篇文章只是做了一个2 * 2 的切割，所以只能在小图片上有很好的效果，但是ViT能够有更大规模的patch，所以可以做到更大规模的图片识别。

ViT的结果

ViT 在中型规模数据集（如：ImageNet）上的识别结果，和同等规模的ResNets相比是要更弱一点的，作者认为的原因在于：Transformer和卷积神经网络相比，缺少一些卷积神经网络所拥有的归纳偏置（inductive biases），可理解为1.在图片上相邻的区域具相邻的特征，2.translation equivariance（平移等边性的原因），如：f（g（x））这个函数，先做f和g其实都是一样的，即无论是先做平移还是先做卷积，最后的结果其实都是一样的，所以不管物体移动到哪里，只要是同样的输入进来，经过同样的卷积核，输出都是一样的。所以基于以上两点，可以理解为，卷积神经网络有更多的先验信息，所以只需要相对少的数据就可以学得一个相对好的模型，而Transformer只能从一点点开始学。当作者在大规模的数据集上去做训练时，就比卷积神经网络的效果要好一些了。

ViT 具体流程



在NLP中，自注意力是两两相互的，但是将图片切割了之后，很显然位置信息就变了，所以图像的位置信息我们通过Linear Projection of Flattened Patches这个来进行全连接生成token，具体操作如下：我们给Patch加上Position Embedding，即加上一个位置编码信息，一旦加上之后，这个token就包含了这个图片原本有的图像信息又包含了这个图像块现在有的位置信息。只要得到每一个patch的token，那么就和NLP中的处理方式一样了，只需要将所有的token输入Transformer Encoder，就会得到相应的很多输出。

当然有这么多的输出token，用哪个输出token去做分类呢?

可以通过加入一个特殊字符的方式，即通过加上Extra learnable embedding字符（class embedding），它也是一个position embedding，其位置信息永远是0。这样的话，所有的token都在相互交互计算，而且这个该class embedding能够做到从别的embedding中去学到有用的信息，所以只需要根据这一个class embedding 去做最后的判断既可以获得结果。

图中的MLP Head是一个通用的分类头，最后用交叉熵函数去做模型的训练。

在ViT中用的Transformer Encoder流程和Transformer中的也是一样的，即Embedding Patch进入Layer Norm以后，进行一系列的Mutil-Head Attention之后，再进入Layer Norm，最后通过MLP，就生成了一个Transformer Block，叠加L次之后就形成了一个Transformer Encoder。

总结

ViT最大的意义在于完全证明了把Transformer用于图像领域是完全可行的，文章里的实验证明了不同情况下的ViT可行度，结论就是在大数据上是比ResNet好的，随着模型的增加，ViT的效果也在不断增加。但是就目前而言，ViT数据需求量大、计算量大、堆叠层数数量受限、模型本身无法编码位置，当然这一年里也有很多优化工作的出现。

未来方向

1. 用ViT去做小样本学习

部分代码：

```
class IdentityLayer(nn.Module):
"""Identity layer, convenient for giving a name to an array."""

@nn.compact
def __call__(self, x):
  return x


class AddPositionEmbs(nn.Module):
"""Adds (optionally learned) positional embeddings to the inputs.


  dtype: the dtype of the computation (default: float32).
  dropout_rate: dropout rate.
  attention_dropout_rate: dropout for attention heads.
  deterministic: bool, deterministic or not (to apply dropout).
  num_heads: Number of heads in nn.MultiHeadDotProductAttention
"""

mlp_dim: int
num_heads: int
dtype: Dtype = jnp.float32
dropout_rate: float = 0.1
attention_dropout_rate: float = 0.1

@nn.compact
def __call__(self, inputs, *, deterministic):
  """Applies Encoder1DBlock module.

  Args:
    inputs: Inputs to the layer.
    deterministic: Dropout will not be applied when set to true.

  Returns:
    output after transformer encoder block.
  """

  # Attention block.
  assert inputs.ndim == 3, f'Expected (batch, seq, hidden) got {inputs.shape}'
  x = nn.LayerNorm(dtype=self.dtype)(inputs)
  x = nn.MultiHeadDotProductAttention(
      dtype=self.dtype,
      kernel_init=nn.initializers.xavier_uniform(),
      broadcast_dropout=False,
      deterministic=deterministic,
      dropout_rate=self.attention_dropout_rate,
      num_heads=self.num_heads)(
          x, x)
  x = nn.Dropout(rate=self.dropout_rate)(x, deterministic=deterministic)
  x = x + inputs

  # MLP block.
  y = nn.LayerNorm(dtype=self.dtype)(x)
  y = MlpBlock(
      mlp_dim=self.mlp_dim, dtype=self.dtype, dropout_rate=self.dropout_rate)(
          y, deterministic=deterministic)

  return x + y


class Encoder(nn.Module):
"""Transformer Model Encoder for sequence to sequence translation.

Attributes:
  num_layers: number of layers
  mlp_dim: dimension of the mlp on top of attention block
  num_heads: Number of heads in nn.MultiHeadDotProductAttention
  dropout_rate: dropout rate.
  attention_dropout_rate: dropout rate in self attention.
"""

num_layers: int
mlp_dim: int
num_heads: int
dropout_rate: float = 0.1
attention_dropout_rate: float = 0.1

@nn.compact
def __call__(self, inputs, *, train):
  """Applies Transformer model on the inputs.
  Args:
    inputs: Inputs to the layer.
    train: Set to `True` when training.

  Returns:
    output of a transformer encoder.
  """
  assert inputs.ndim == 3 # (batch, len, emb)

  x = AddPositionEmbs(
      posemb_init=nn.initializers.normal(stddev=0.02), # from BERT.
      name='posembed_input')(
          inputs)
  x = nn.Dropout(rate=self.dropout_rate)(x, deterministic=not train)

  # Input Encoder
  for lyr in range(self.num_layers):
    x = Encoder1DBlock(
        mlp_dim=self.mlp_dim,
        dropout_rate=self.dropout_rate,
        attention_dropout_rate=self.attention_dropout_rate,
        name=f'encoderblock_{lyr}',
        num_heads=self.num_heads)(
            x, deterministic=not train)
  encoded = nn.LayerNorm(name='encoder_norm')(x)

  return encoded


class VisionTransformer(nn.Module):
"""VisionTransformer."""

num_classes: int
patches: Any
transformer: Any
hidden_size: int
resnet: Optional[Any] = None
representation_size: Optional[int] = None
classifier: str = 'token'

@nn.compact
def __call__(self, inputs, *, train):

  x = inputs
  # (Possibly partial) ResNet root.
  if self.resnet is not None:
    width = int(64 * self.resnet.width_factor)

    # Root block.
    x = models_resnet.StdConv(
        features=width,
        kernel_size=(7, 7),
        strides=(2, 2),
        use_bias=False,
        name='conv_root')(
            x)
    x = nn.GroupNorm(name='gn_root')(x)
    x = nn.relu(x)
    x = nn.max_pool(x, window_shape=(3, 3), strides=(2, 2), padding='SAME')

    # ResNet stages.
    if self.resnet.num_layers:
      x = models_resnet.ResNetStage(
          block_size=self.resnet.num_layers[0],
          nout=width,
          first_stride=(1, 1),
          name='block1')(
              x)
      for i, block_size in enumerate(self.resnet.num_layers[1:], 1):
        x = models_resnet.ResNetStage(
            block_size=block_size,
            nout=width * 2**i,
            first_stride=(2, 2),
            name=f'block{i + 1}')(
                x)

  n, h, w, c = x.shape
```

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], Lucas Beyer[*], Alexander Kolesnikov[*], Dirk Weissenborn[*], Xiaohua Zhai[*], Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

1 INTRODUCTION

Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP). The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019). Thanks to Transformers' computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters (Brown et al., 2020; Lepikhin et al., 2020). With the models and datasets growing, there is still no sign of saturating performance.

In computer vision, however, convolutional architectures remain dominant (LeCun et al., 1989; Krizhevsky et al., 2012; He et al., 2016). Inspired by NLP successes, multiple works try combining CNN-like architectures with self-attention (Wang et al., 2018; Carion et al., 2020), some replacing the convolutions entirely (Ramachandran et al., 2019; Wang et al., 2020a). The latter models, while theoretically efficient, have not yet been scaled effectively on modern hardware accelerators due to the use of specialized attention patterns. Therefore, in large-scale image recognition, classic ResNet-like architectures are still state of the art (Mahajan et al., 2018; Xie et al., 2020; Kolesnikov et al., 2020).

Inspired by the Transformer scaling successes in NLP, we experiment with applying a standard Transformer directly to images, with the fewest possible modifications. To do so, we split an image into patches and provide the sequence of linear embeddings of these patches as an input to a Transformer. Image patches are treated the same way as tokens (words) in an NLP application. We train the model on image classification in supervised fashion.

When trained on mid-sized datasets such as ImageNet without strong regularization, these models yield modest accuracies of a few percentage points below ResNets of comparable size. This seemingly discouraging outcome may be expected: Transformers lack some of the inductive biases

[1]Fine-tuning code and pre-trained models are available at `https://github.com/google-research/vision_transformer`