

# HW-9-陈宸

## Q1.ViT代码理解

### 1.Explain TODO in chunking comment

`einops.rearrange()`操作把原本的图片分割成`tile_size*tile_size`个图像块，后将每个图片转换成一维的向量，故`tiles[0,5,0,:]`表示第一个批次中的第六个小块，该分块对应于原始图像中从行号7到14，列号7到14的区域，也就是第二行第二列的小块，与转换后的第六个小块是等价的。

### 2.TODO: explain what is the value of context\_size and its meaning

`img_size/tile_size`得到图像沿一个方向被分割成的份数，平方后得到总分割块数。为实现对图像进行类别的判断，ViT引入类似flag的class token，其输出特征加上线性分类器即可实现分类。在训练过程前，class token随机初始化embedding并加在被分割后图片序列的0号位置处，与原始序列一同输入transformer，故`context_size`需要加一。

### 3.Refer to Transformer paper and explain the function of the position encoding

在该函数中首先调整了正余弦函数的频率，对输入的每一行数据，将偶数下标的位置使用正弦函数进行编码，奇数位置采用余弦函数进行位置编码，以引入位置信息，使得transformer能够区分不同位置的输入。

### 4.Explain each step in MultiheadAttention forward function and associate with the QKV equation

**类定义部分：**定义了多头注意力机制所需的各种参数和层。其中，`self.qkv`是一个线性层，用于同时生成查询（Query）、键（Key）和值（Value）。我们将查询、键和值的嵌入维度都设置为`embed_size`的三倍，以便在后续的计算中能够将其分割为三个部分。

#### **forward()工作：**

- 1.首先执行层归一化，后通过线性层对输入的 $x$ 生成查询 $q$ ， $k$ ， $v$ ，然后沿最后一个维度将结果分成三部分。通过`view`和`transpose`将每个头的维度分离，使每个头的维度为`self.embed_size // self.n_heads`，即将嵌入向量分成多个头以便并行计算。
- 2.计算查询 $q$ 和键 $k$ 的点积，得到相关性矩阵。
- 3.缩放相关性矩阵。
- 4.对相关性矩阵进行softmax操作，将值归一化到0到1之间。
- 5.应用Dropout。
- 6.计算加权和值，即注意力机制的输出。
- 7.将多头注意力的输出重新组合为原来的形状。
- 8.最后再应用一次Dropout，返回最终的输出 $y$ 。

结合多头注意力机制中QKV的公式：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

association:

```
self.qkv(x).chunk(3, dim=-1)           //实现了生成Q, K, V的功能
correlation = q @ k.transpose(-2, -1)    //实现了Q和K的点积, 计算相关性矩阵
correlation = correlation / math.sqrt(k.shape[-1])
F.softmax(correlation, dim=-1)           //实现了缩放和softmax操作
y = correlation @ v                       //实现了对V的加权求合值
```

##### 5.Explain why self.qkv is one linear function instead of three

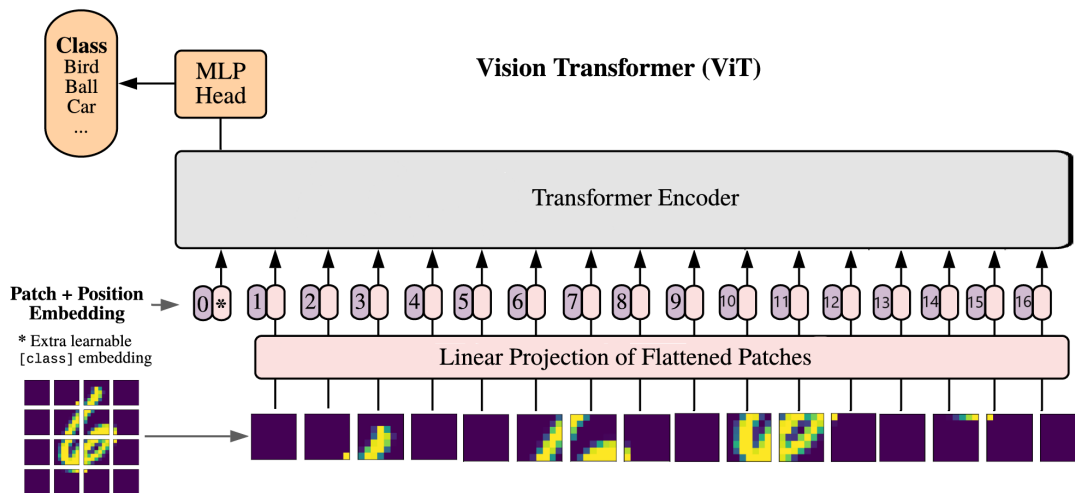
self.qkv 作为一个线性层可以一次性生成Query、Key和Value, 相对于使用三个线性层可以减少计算量, 提高效率。

##### 6.Explain why q,k,v transpose(1,2) and y transpose back

结合代码, 发现各个头部的注意力输出维数为 (Batch, Num Heads, Sequence Length, Head Dim)。

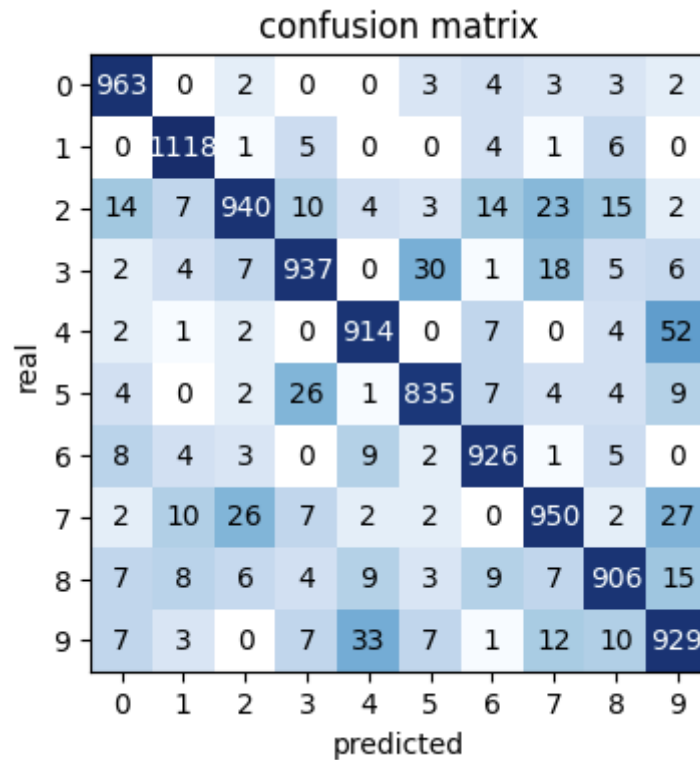
而合并多个头的步骤中, 需要将维度最后输出为(Batch, Sequence Length, Embedding Size), 故需要先将维度转置回来, 使其形状变为 (Batch, Sequence Length, Num Heads, Head Dim), 然后通过 view 方法重新调整形状为能够连接回原始的嵌入维度。

outline of ViT:



##### 7.We want to know which classes are misclassified the most.Provide top-5 cases, that True class A is misclassified as B. Print A->B

根据输出的混淆矩阵, 4->9, 9->4, 3->5, 7->9, 7->2为分类错误率最高的5个分类序列。



#### 8.Explain what is the size and functionality of tile\_embedding layer

input\_size:TILE\_SIZE \* TILE\_SIZE \* CHANNEL

output\_size:embed\_size

##### Functionality:

tile\_embedding层用于将输入数据从一个形状转换为另一个形状，并将其嵌入到更高维的空间中，通过变换，模型可以在更高维空间中对不同位置的信息进行更好地区分和处理。

#### Visualize attention

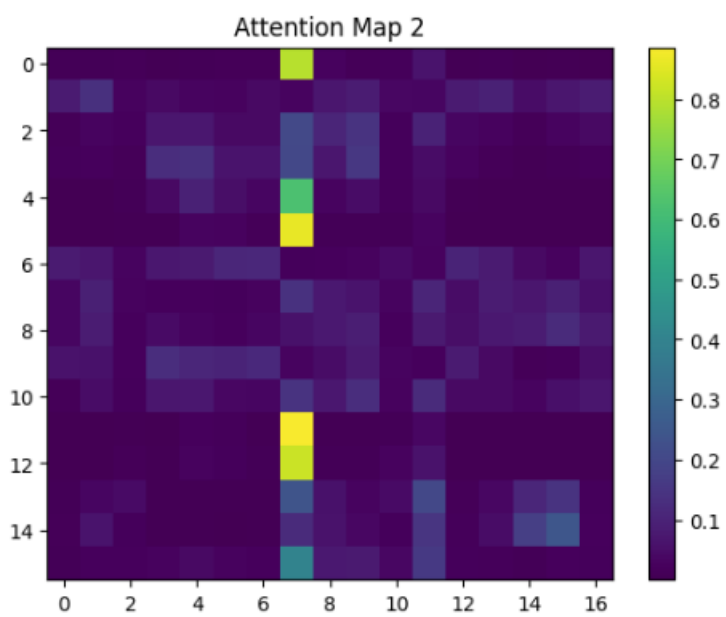
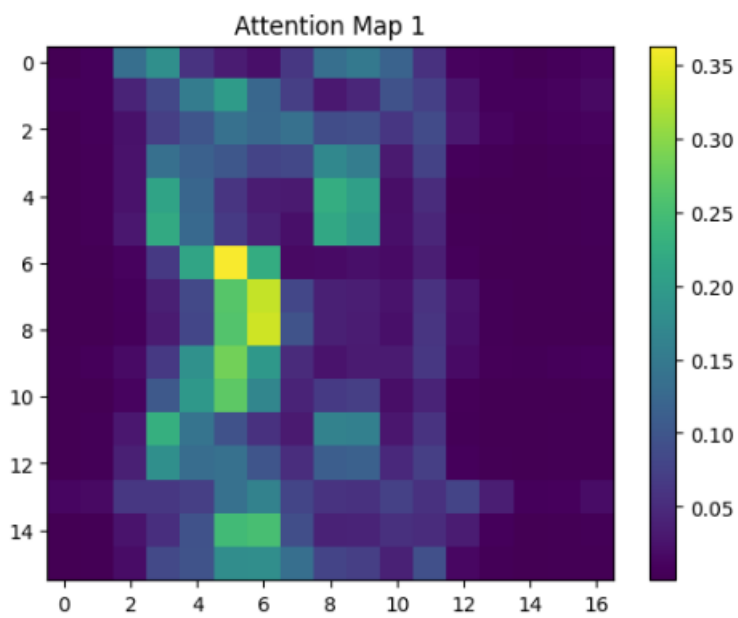
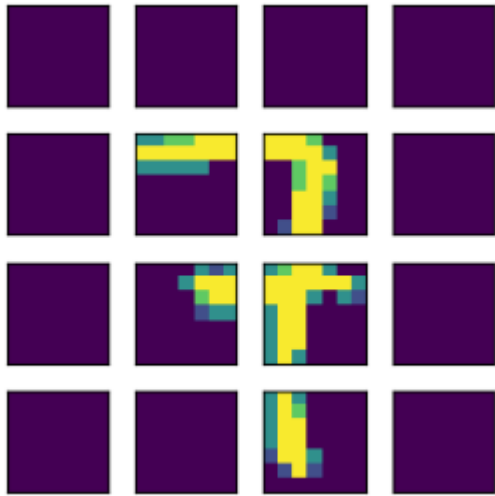
修改可视化attention map代码如下（所给代码存在act维数与plt.imshow()要求不一致的错误）

```
@torch.no_grad()
def visualize_attention(model, img):
    model.eval()
    plt.figure()
    plt.imshow(img.permute(1, 2, 0)) # 展示原始图片
    tiles = einops.rearrange(img, 'c (h t1) (w t2) -> (h w) c t1 t2', t1=TILE_SIZE,
t2=TILE_SIZE)
    plot(tiles, n_col=4, block_size=0.75)
    img = img.to(device).unsqueeze(0)
    out, activations = model.forward_visualize(img)
    out = out.argmax(-1).item()

    for block, acts in enumerate(activations):
        acts = acts.squeeze()
        acts = acts[1:, 1:].cpu()
        if acts.ndim == 3 and acts.shape[0] == 1:
            plt.figure()
            plt.imshow(acts.squeeze(0), cmap='viridis')
            plt.gca().axes.xaxis.set_major_locator(MaxNLocator(integer=True))
```

```
plt.title(f'Attention Map {block+1}')
plt.colorbar()

visualize_attention(model, next(iter(data))[0][0])
```



根据绘制出的attention map，颜色较亮的部分说明横纵坐标对应的token编号相关性较强，同时为模型贡献了较高的注意力权重，模型对这些区域的关注度较高。

根据attention map1，token编号6与5、7、8关系较为紧密，均为亮色块或亮色边界，属于模型能够理解的图片的浅层的特征。

根据attention map2，token编号7与11、12、5关系比较紧密，均为暗色块或暗色边界。属于模型理解的图片的较为深层的特征。

## Q2.RoPE位置编码

### RoPE作用于每一层QK上的原因：

首先，在注意力机制中，主要是通过计算Q和K之间的相似度来决定注意力权重，而RoPE的作用于每一层可以使模型能够理解每一层的位置信息，增强了其对于局部特征的关注能力，同时确保模型在多层处理中不会丢失关于位置的重要信息。

并且作用在每一层上Q，K的RoPE可以利用参数共享来减少模型的参数量，同时保持模型的灵活性。每一层都能够独立地学习不同的位置编码，而不会因为在输入x上应用而导致位置编码在不同层之间共享。

参考多头注意力机制类编写RoPE类代码如下：

```
class RoPEAttention(nn.Module):
    def __init__(self, n_heads, embed_size, dropout, n_block=0):
        super().__init__()
        self.n_heads = n_heads
        self.embed_size = embed_size
        self.dropout_rate = dropout
        self.qkv = nn.Linear(embed_size, embed_size * 3, bias=False)
        self.dropout = nn.Dropout(dropout)
        self.ln = nn.LayerNorm(embed_size)

    def forward(self, x):
        B, C, E = x.shape
        x = self.ln(x)
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        q = self.apply_rope(q)
        k = self.apply_rope(k)
        q = q.view(B, C, self.n_heads, self.embed_size // self.n_heads).transpose(1, 2)
        k = k.view(B, C, self.n_heads, self.embed_size // self.n_heads).transpose(1, 2)
        v = v.view(B, C, self.n_heads, self.embed_size // self.n_heads).transpose(1, 2)
        correlation = q @ k.transpose(-2, -1) / math.sqrt(k.shape[-1])
        correlation = torch.nn.functional.softmax(correlation, dim=-1)
        correlation = self.dropout(correlation)
        y = correlation @ v
        y = y.transpose(1, 2).contiguous().view(B, C, self.embed_size)
        y = self.dropout(y)
        return y

    def apply_rope(self, x):
        seq_len, dim = x.shape[-2:]
        position = torch.arange(0, seq_len, dtype=torch.float).unsqueeze(-1)
        div_term = torch.exp(torch.arange(0, dim, 2, dtype=torch.float) * -(math.log(10000.0) /
dim))
        pe = torch.zeros(seq_len, dim, device=x.device)
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        return x * pe
```

```

class Block(nn.Module):
    def __init__(self, n_heads, embed_size, hidden_size, dropout, n_block):
        super().__init__()
        self.block = n_block
        self.attention = RoPEAttention(n_heads, embed_size, dropout=dropout, n_block=n_block)
        self.ff = nn.Sequential(
            nn.LayerNorm(embed_size),
            nn.Linear(embed_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, embed_size),
            nn.Dropout(dropout)
        )

    def forward(self, x):
        x = x + self.attention(x)
        x = x + self.ff(x)
        return x

    def forward_visualize(self, x):
        x1, cor = self.attention.forward_visualize(x)
        x = x + x1
        x = x + self.ff(x)
        return x, cor

```

在Net中实现如下:

```


class Net(nn.Module):
    def __init__(self, n_heads=N_HEADS, n_blocks=N_BLOCKS, embed_size=EMBED_SIZE,
                 hidden_size=HIDDEN_SIZE, vocab_size=VOCAB_SIZE, context_size=CONTEXT_SIZE, dropout=DROPOUT,
                 tile_size=TILE_SIZE):
        super().__init__()
        self.context_size = context_size
        self.tile_size = tile_size
        self.tile_embedding = nn.Linear(TILE_SIZE * TILE_SIZE * CHANNEL, embed_size)
        self.cls_token = nn.Parameter(torch.randn(1, 1, embed_size))
        self.blocks = nn.Sequential(*[Block(n_heads, embed_size, hidden_size, dropout, i) for i in
range(n_blocks)])
        self.head = nn.Linear(embed_size, vocab_size)


    def forward(self, x):
        x = einops.rearrange(x, 'b c (h t1) (w t2) -> b (h w) (c t1 t2)', t1=self.tile_size,
t2=self.tile_size)
        x = self.tile_embedding(x)
        cls_token = self.cls_token.expand(x.shape[0], -1, -1)
        x = torch.cat((cls_token, x), dim=1)
        x = self.blocks(x)
        x = self.head(x)
        x = x[:, 0, :]
        return x

```


30epoch结果比对:


get\_sinusoidal\_positional\_encoding():

```
 # test set accuracy  
_ = accuracy(model)  
# train set accuracy  
_ = accuracy(model, label='train')
```

```
 accuracy on test : 0.9217000007629395  
accuracy on train: 0.9210667014122009
```

RoPE-Attention:

```
 # test set accuracy  
_ = accuracy(model)  
# train set accuracy  
_ = accuracy(model, label='train')
```

```
 accuracy on test : 0.9369999766349792  
accuracy on train: 0.9391500353813171
```

使用旋转位置编码的效果略优于相对位置编码。