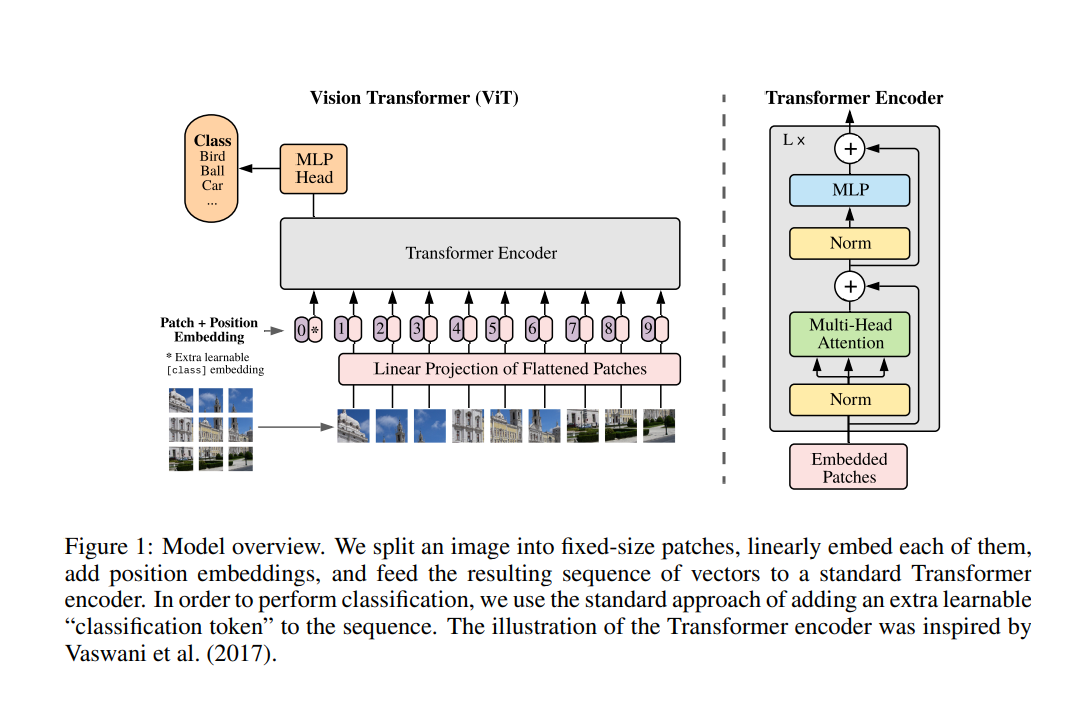
任务：图片分类

使用的数据集：[Classify Leaves | Kaggle](https://www.kaggle.com/c/classify-leaves)

使用的模型：简单的Vision Transformer模型

模型介绍：

Vision Transformer是Transformer在视觉上应用的成果之一，采用Transformer编码器对图片进行特征提取和融合。ViT将图片截取成一批固定大小的图片，resize成向量、附加上位置编码与一个用于采集信息的分类向量，后作为输入进入编码器。最后将分类向量作为多层感知机的输入进行分类。



模型实现：

多层感知机：

使用GELU作为激活函数，使用了Dropout，代码如下

class FeedForward(nn.Module):

def \_\_init\_\_(self, dim, hidden\_dim, dropout=0.):

super().\_\_init\_\_()

self.net = nn.Sequential(

nn.Linear(dim, hidden\_dim),

nn.GELU(),

nn.Dropout(dropout),

nn.Linear(hidden\_dim, dim),

nn.Dropout(dropout)

)

def forward(self, x):

return self.net(x)

正则化：

使用LayerNorm而非BatchNorm，代码如下：

class PreNorm(nn.Module):

def \_\_init\_\_(self, dim, fn):

super().\_\_init\_\_()

self.norm = nn.LayerNorm(dim)

self.fn = fn

def forward(self, x, \*\*kwargs):

return self.fn(self.norm(x), \*\*kwargs)

自注意力：

使用了多头注意力，将单个向量的不同部位分离来探索不同的隐空间，使用einsum库辅助了attention\_score的计算，使用了点积计算注意力，使用单个前馈层对KQV进行处理，关键代码如下：

def forward(self, x):

b, n, \_, h = \*x.shape, self.heads

qkv = self.to\_qkv(x).chunk(3, dim=-1)

q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h=h), qkv)

dots = einsum('b h i d, b h j d -> b h i j', q, k) \* self.scale

attn = self.attend(dots)

out = einsum('b h i j, b h j d -> b h i d', attn, v)

out = rearrange(out, 'b h n d -> b n (h d)')

return self.to\_out(out)

Transformer编码器：

参考了上图结构，构建了Transformer编码器，具体结构见下图代码：

class Transformer(nn.Module):

def \_\_init\_\_(self, dim, depth, heads, dim\_head, mlp\_dim, dropout=0.):

super().\_\_init\_\_()

self.layers = nn.ModuleList([])

for \_ in range(depth):

self.layers.append(nn.ModuleList([

PreNorm(dim, Attention(dim, heads=heads,

dim\_head=dim\_head, dropout=dropout)),

PreNorm(dim, FeedForward(dim, mlp\_dim, dropout=dropout))

]))

def forward(self, x):

for attn, ff in self.layers:

x = attn(x) + x

x = ff(x) + x

return x

Vision Transformer：

采用如上组件构建了Vision Transformer模型，除了上述组件外，Vision Transformer类的工作还有：

（1），将图片分割为Patch，并通过前馈层减少计算复杂度，代码如下：

nn.Sequential(

Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)',

p1=patch\_height, p2=patch\_width),

nn.Linear(patch\_dim, dim))

（2）、添加位置嵌入向量与分类嵌入向量，具体代码如下：

# b表示batchSize, n表示每个块的空间分辨率, \_表示一个块内有多少个值

b, n, \_ = x.shape

# self.cls\_token: (1, 1, dim) -> cls\_tokens: (batchSize, 1, dim)

cls\_tokens = repeat(self.cls\_token, '() n d -> b n d', b=b)

# 将cls\_token拼接到patch token中去

x = torch.cat((cls\_tokens, x), dim=1)

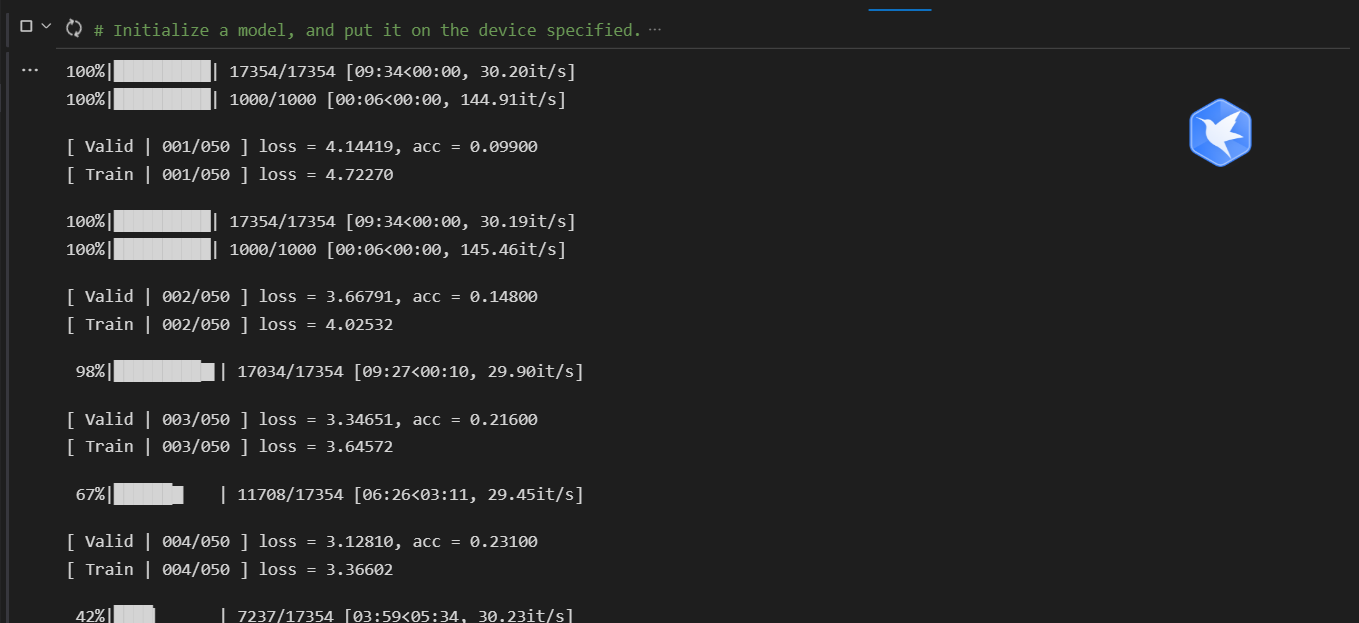
x += self.pos\_embedding[:, :(n+1)]

最后，将x输入到Transformer中进行训练即可。

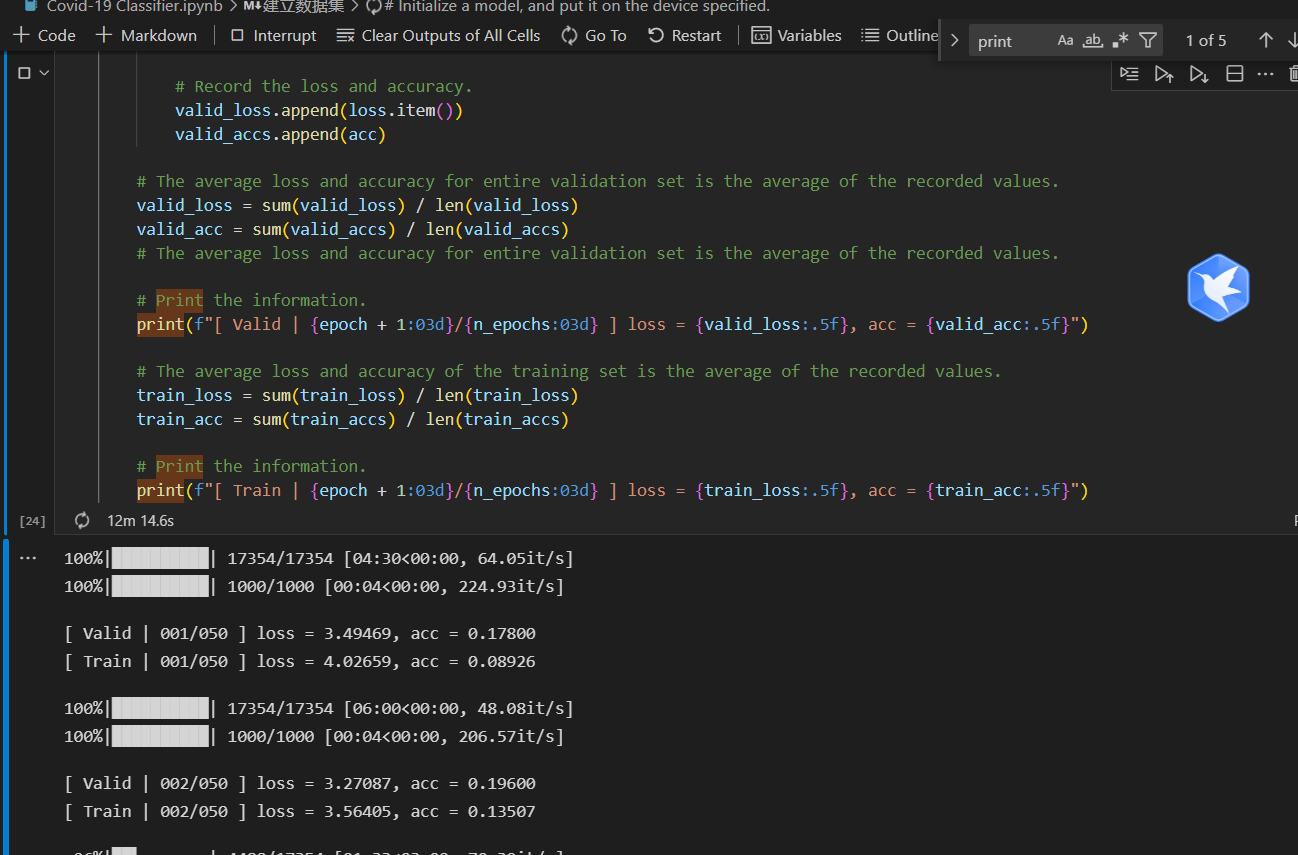
训练参数：

使用了一定的数据增强。图片尺寸为224x224，patch大小为32x32,使用了六层Transformer编码器，16个注意力头，多层感知机的隐藏层大小为2048个神经元。

训练过程：



由于模型复杂度过高，拟合效果不好，调整了编码器层数。



更换编码器层数后拟合效果仍然不好，预计是数据过少导致。