烟台大学课程论文

课程名称： 机器学习

论文题目：基于Swin Transformer人脸表情识别系统

院（部） 名 称： 计算机与控制工程学院

指导老师姓名： 刘志中

设计时间： 2021.12.23

课程论文成绩评定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **评定内容** | **评定指标** | | **分值** | **成 绩** |
| **工作态度** | **态度认真；严谨务实； 能够按期完成规定的内容；** | | 15 |  |
| **论文格式** | **语言通畅；格式规范；结构合理；内容完整** | | 25 |  |
| **论文质量** | **内容全面；表述清晰；模型正确；严谨合理；技术用语准确；符号统一；书写工整规范；图表完备、整洁、正确；** | | 30 |  |
| **工作创新** | **论文中有一定的创新意识、有自己的贡献；**  **对前人工作有改进或突破，或有独特见解；** | | 10 |  |
| **量与难度** | **工作量饱满，工作难度大** | | 20 |  |
| **总成绩：** | | | | |
| **指导老师签字：** | | **论文完成日期：**  2021.12.23 | | |
| **评定日期：** | | |

目录

[摘要 1](#_Toc9958)

[一、课程论文目的： 1](#_Toc10898)

[二、总体思路 1](#_Toc12099)

[三、 实现过程 6](#_Toc9543)

[3.1数据预处理 6](#_Toc23910)

[3.2网络的构建 7](#_Toc4401)

[3.3网络训练与测试： 7](#_Toc2872)

[3.4 项目结果展示: 11](#_Toc2720)

[四、总结 13](#_Toc1952)

[五、课程感想与建议 14](#_Toc5094)

[参考文献 17](#_Toc18032)

[附录 18](#_Toc17709)

摘要

本文的主要工作内容是设计一个人脸表情识别系统，能够对日常生活中的基本表情进行识别。在模型选取方面，我们选取了荣获CVPR2021最佳论文的Swin Transformer模型，该模型通过引入NLP领域中的注意力机制和Transformer机制，使得该模型相比于CNN模型有了较大的准确率提升；另外我们还设计了一个Web界面，用户可以上传人脸图片，系统会将识别结果显示在前端。

一、课程论文目的：

设计一个人脸表情识别系统，可以对基本的人脸表情进行识别；同时提供一个可视化的操作界面，可以使用训练好的模型对图片进行检测，向用户及时提供模型识别结果，或者属于某个表情的概率，并且按照概率从大到小展示。

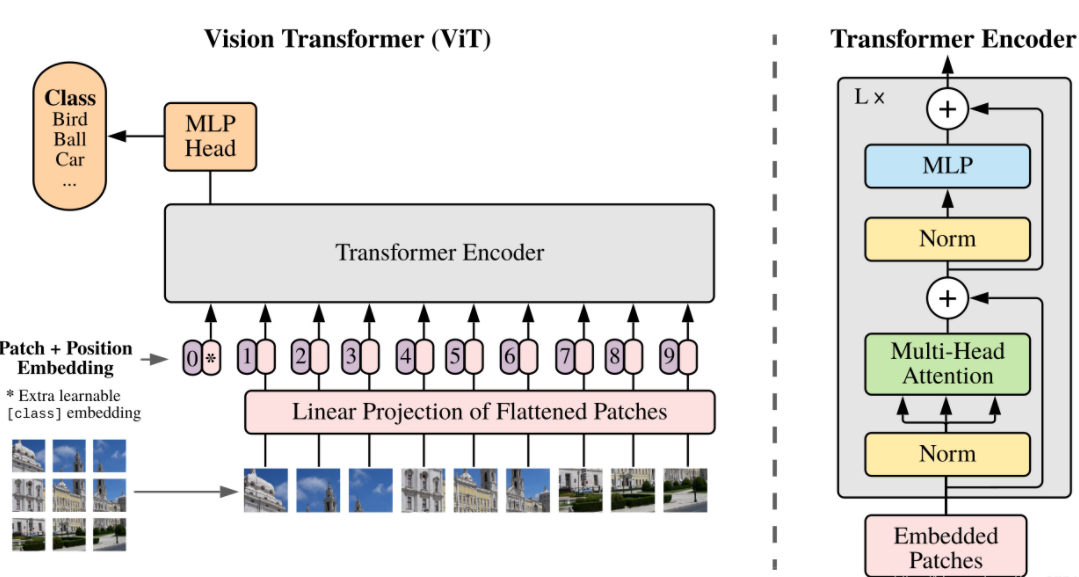
二、总体思路

人脸表情识别系统本质是一个图像分类的问题，类似于实验8做的Minst手写体分类。在机器学习中分类问题的本质是学习一个条件概率分布，通过该分布去给出图像属于某一类别的概率，选取类别概率最大的类别作为该图像最后的类别预测；传统的做法是使用CNN网络对图像进行特征提取，最后通过全连接层进行分类，使用SoftMax操作进行归一化，使其变成一个概率分布。

传统的CNN做法在提取图像特征时存在一个不足，首先是CNN的空间不变性，卷积核在所有的空间位置上都共享参数，那就会导致不同空间位置的局部空间建模能力受限，并无法有效的捕获空间上长距离的关系。具体来说卷积核会遍历整个图像，对于图像的每一个卷积窗口使用的是相同的卷积核，这也是CNN权重共享的表现之一，这些的做法会导致一些问题，就是说对于一幅图像来说，CNN并不会去关注某些特别的区域，对于每一个卷积窗口都是使用相同的提取方式，这对于图像的特征提取来说不是一个很好的方面；另一个方面在CNN的通道特异性，在深层CNN网络中会不断放大图像的通道数，以此来获取不同方面或角度的特征信息，但是每个通道之间是相互独立的，它们之间缺乏信息交流，可能会出现信息冗余的情况、参数多和计算量大。

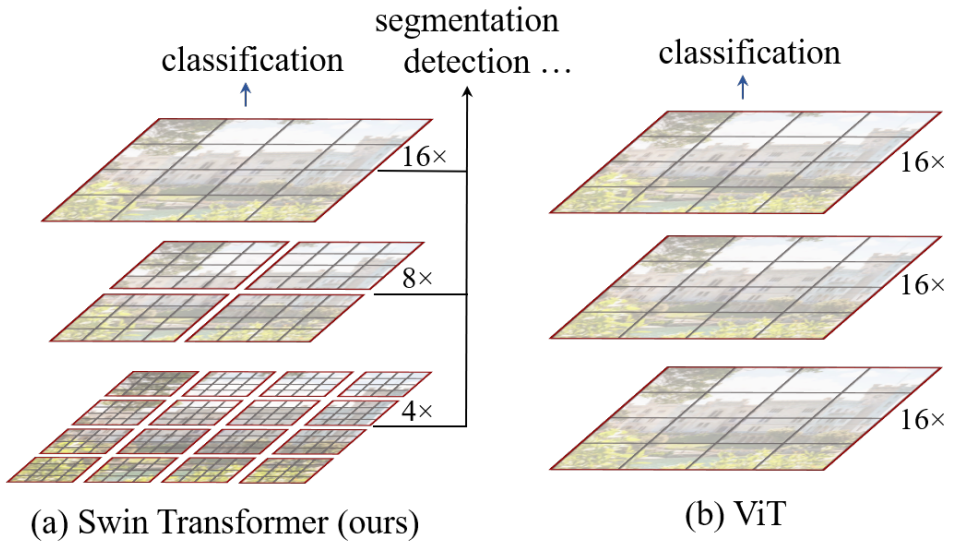
针对这些情况，近些年来有很多学者把NLP领域中的Transformer引入到了CV领域，进而出现来Vision Transformer网络(网络结构见图1)，这种结构首先通过一个Embedding层对图像进行编码(实际操作中是使用CNN对图像进行降采样然后拉平)，形成一堆token，然后在加入Position Embedding编码，使得序列含有位置信息，然后再加入一个class token，作为分类的token，然后送入Transformer网络，在Transformer网络中会通过一个多头注意力层和MLP层，在多头注意力层中通过注意力机制可以使得网络关注图像中一些特别的区域，降低其余无用区域的关注度；网络的最后是一个MLP Head，将class token提取出来，然后送入MLP网络进行分类。

ViT模型将Transformer引入，确实达到了很好的效果，但依然存在一些不足，ViT网络缺乏CNN的层次化结构，就说CNN网络使用不同大小的卷积核，一般来说感受野随着深度加深逐渐加大，不同层次可以看到不一样的信息，有更好的学习能力，这样也会导致ViT训练的时下游任务收敛的慢，因此需要很多数据才能到达一个不错的效果。



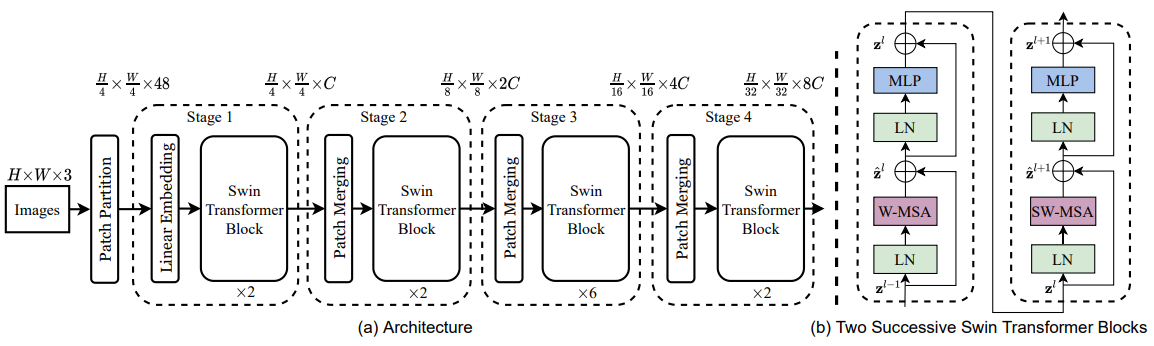
**图 1**

随后就出现了本文所使用的Swin Transformer网络，首先解决的一个问题就是说层次化结构，在ViT中开始对feature map使用16倍下采样，后面的网络层也是维持这个下采样率不变，而Swin Transformer则对feature map使用例如4倍、8倍和16倍这样的下采样率(见图2)，类似于CNN中的层次化结构；另一个就是Swin Transformer引入Window Attention机制，避免对整个feature map做Self-Attention，可以大大降低计算量，使得计算复杂图和图像大小呈线性关系。



**图 2**

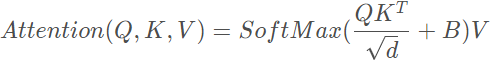
Swin Transformer整体网络结构见下图3：



**图 3**

这里假设图像是三通道图像，首先通过Patch Partition对图像进行下采样(这里的patch size是4)，通过设置一个卷积核和步长都设置为patch size来实现下采样，然后将feature map拉直，再通过Linear Embedding层进行Layer Norm，然后将通道数调整为Embedding Dim，Patch Partition与Linear Embedding共同组成了Patch Embedding层，将图像编码成序列，这这个地方与ViT不同的是Position Embedding编码是可选的，因为在后续Block中有一个Mask相对编码。

每一个Swin Transformer Block都由两个基本模块组成(图3右边)，两个模块都是由窗口注意力和MLP层组成，但是两者的窗口注意力层是不一样的，前者的窗口大小是固定的，后者是移位窗口注意力，不同的窗口大小不一样。具体来说在W-MSA中，先对图像按窗口大小划分得到一系列不重叠窗口，然后再在这些窗口上做Self-Attention。在每个窗口上分别做Self-Attention显然比在整个feature map上做的计算量要小很多，但是如果仅仅是在单个窗口上做Self-Attention也会有一些问题就是说窗口之间缺乏信息交流，因此作者引入窗口注意力机制来解决这一个问题，移位窗口注意力机制首先对将feature map划分为不同大小的不重叠的窗口，然后将这些窗口进位偏移得到一个新的feature map，再在这个feature map上做固定窗口的Self-Attention，这些做会产生另外一个问题，由于偏移后的feature map有些窗口中的像素本来是不相邻，如果直接进行Self-Attention显然是不合理的，因此作者引入一个Mask操作来屏蔽那些位置不相邻的像素点，具体来说是通过在Attention计算公式中引入一个Mask矩阵B，对于那些不相邻的像素，通过加一个较大的负值(比如-100.0)，使得通过SoftMax操作后的值趋于0来达到屏蔽的效果。



在进行完移位窗口注意力后需要恢复成移位前的状态。Mask如何判断像素相邻是通过相对位置生成的，对于每个像素点计算一次图中其他像素的相对坐标，然后拉直，然后再加一个偏移值，使得坐标值最小为0，然后通过2\*x+y操作将二元坐标转换成单一的索引值，然后同查询相对位置表来给每个位置赋一个值，这个相对位置表是一个可学习的参数，最后通过判断每个像素的值是否相同来判断是否相邻的像素点。

相邻的Swin Transformer Block的feature map都进行了高宽缩小一倍，通道数扩大一部的操作，这是通过Patch Merging层完成的，Patch Merging首先对每个feature map按窗口大小进行分割，然后在通道方向展开，然后在经过一个LN进行归一化，最后通过全连接层将通道数调整为之前的两倍。

对于分类模型来说，在最后使用全局自适应平局池化层将feature map展平，然后使用一个全连接层进行映射，通过SoftMax将结果进行归一化，得到一个各个类别的概率分布，获取预测结果。

1. 实现过程

3.1数据预处理

使用torchvision的ImageFolder对本地图像文件进行加载，并使用torchvison的transforms将图片转成tensor格式。

数据：使用CK+48数据，一共有7种表情类型

# 预处理并加载数据

def load\_dataset(filePath, batch\_sz=128, val\_sz=0.4):

trans = transforms.Compose([transforms.ToTensor()])

dataset = ImageFolder(root=filePath,

transform=trans)

data\_len = len(dataset)

val\_len = int(data\_len \* val\_sz)

train\_len = data\_len - val\_len

train\_dataset, val\_dataset = random\_split(dataset=dataset,

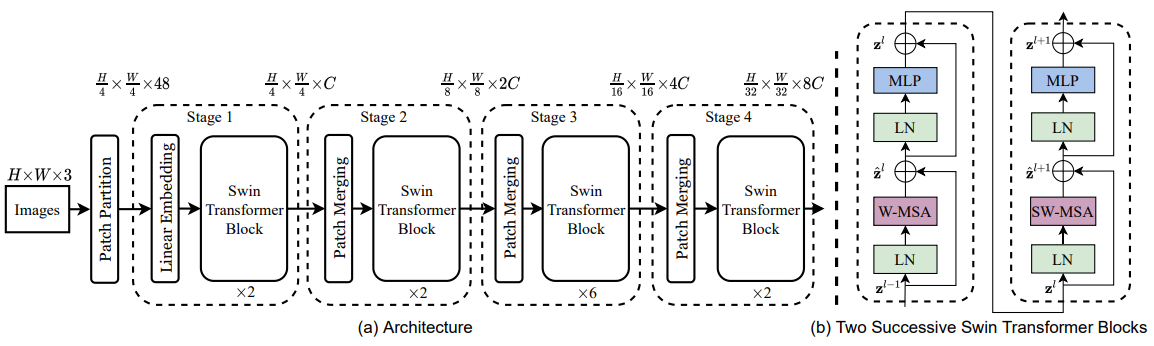
lengths=[train\_len, val\_len])

return (DataLoader(train\_dataset, batch\_size=batch\_sz, shuffle=True),

DataLoader(val\_dataset, batch\_size=batch\_sz, shuffle=True),

dataset.class\_to\_idx)

3.2网络的构建



项目参数:

Swin Transformer参数:

patch\_size =2,window\_size=5,embed\_dim=96

训练参数：AdamW和SGD训练器，学习率分别为0.0001, 0.001，训练轮数分别为108和525次，AdamW收敛的更快。

结果展示：Python Flask框架搭建了一个简易的Web服务器，前端发来照片通过模型进行推理，将推理后的结果发送到前端展示。

3.3网络训练与测试：

训练测试主要代码:

# 每一个训练epoch

def train\_epoch(model, train\_loader, optimizer, loss\_fn, epoch):

size = len(train\_loader.dataset)

num\_batches = len(train\_loader)

model.train() # 将模型设置为训练模式

train\_loss, train\_correct = 0, 0

for batch\_idx, (X, y) in enumerate(train\_loader):

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

# 梯度清零， 反向传播，更新网络参数

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 记录损失与正确率

train\_loss += loss.item()

train\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(X), size,

100. \* batch\_idx / num\_batches, loss.item()))

return train\_loss / num\_batches, train\_correct / size

# 测试epoch

def test\_epoch(model, test\_loader, loss\_fn):

size = len(test\_loader.dataset)

num\_batches = len(test\_loader)

model.eval() # 设为评估模式

test\_loss, test\_correct = 0, 0

# 不记录梯度，节省内存

with torch.no\_grad():

for X, y in test\_loader:

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

test\_loss += loss.item()

test\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

test\_loss /= num\_batches

print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, test\_correct, size, 100. \* test\_correct / size))

return test\_loss, test\_correct / size

# 分类问题使用交叉熵作为损失函数

loss\_fn = nn.CrossEntropyLoss()

# 使用SGD或者AdamW进行训练

trainer = torch.optim.AdamW(net.parameters(), lr = 0.0001, weight\_decay=5E-2)

# 使用DP模式训练

net = nn.DataParallel(net)

# 获取训练数据集和测试数据集

# 训练轮数

num\_epochs = 500

# 记录损失和正确率

train\_loss, train\_accuracy = [], []

test\_loss, test\_accuracy = [], []

for epoch in range(1, num\_epochs + 1):

print(f"Epoch {epoch}\n-------------------------------")

a, b = train\_epoch(net, train\_iter, trainer, loss\_fn, epoch)

train\_loss.append(a)

train\_accuracy.append(b)

c, d = test\_epoch(net, test\_iter, loss\_fn)

test\_loss.append(c)

test\_accuracy.append(d)

writer.add\_scalar('train/loss',scalar\_value= a, global\_step=epoch)

writer.add\_scalar('train/Accuracy', scalar\_value=b, global\_step=epoch)

writer.add\_scalar('test/loss', scalar\_value=c, global\_step=epoch)

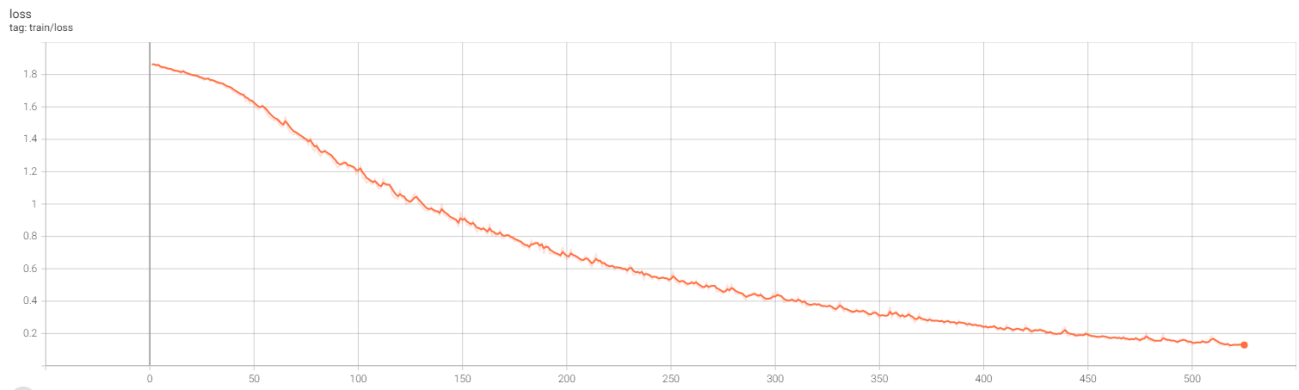
writer.add\_scalar('test/Accuracy', scalar\_value=d,global\_step=epoch)

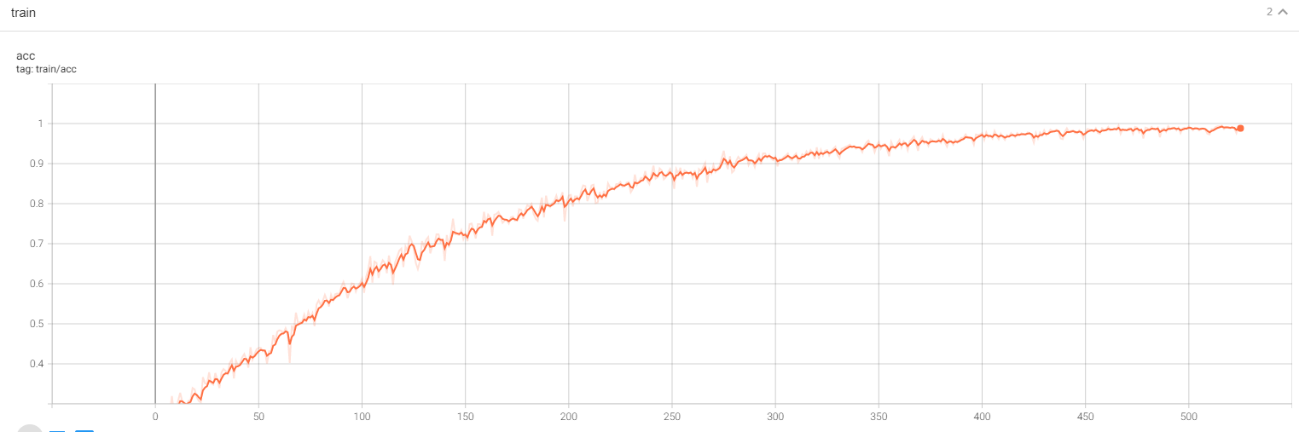
# 保存模型参数

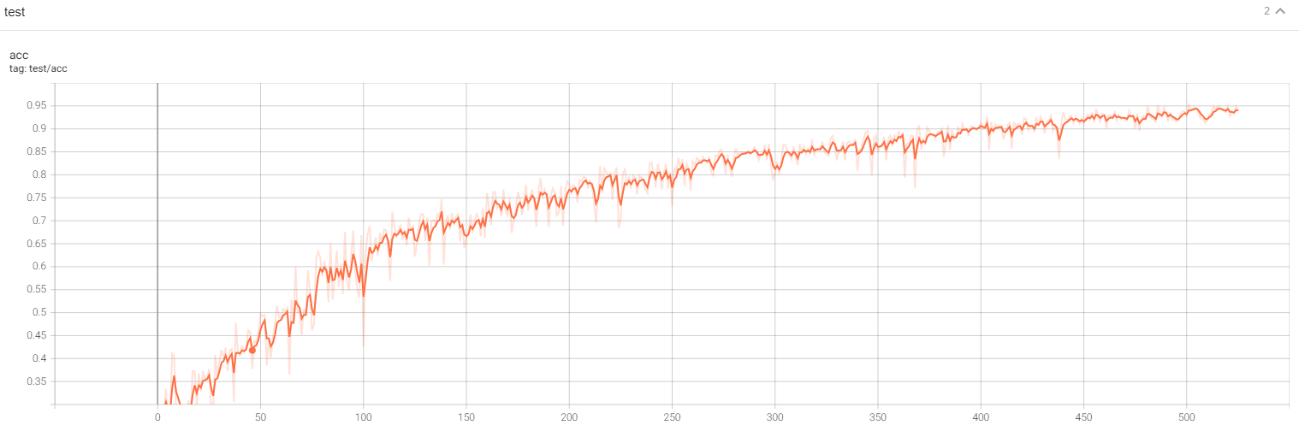
torch.save(net.state\_dict(),'./model.pth')

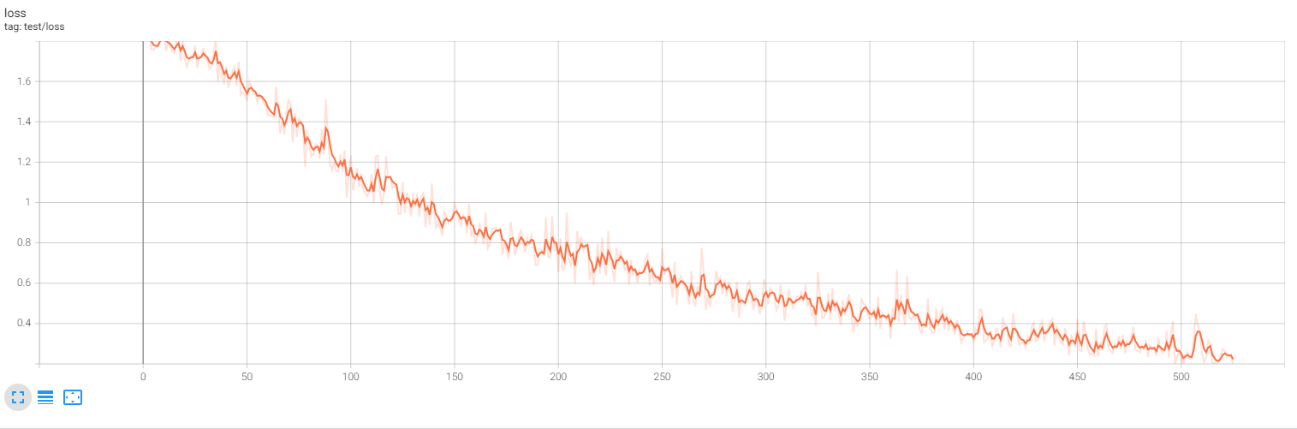
训练测试结果：

使用SGD训练525轮(95%准确率)，从上到下分别是训练损失，训练精度，测试精度，测试损失：

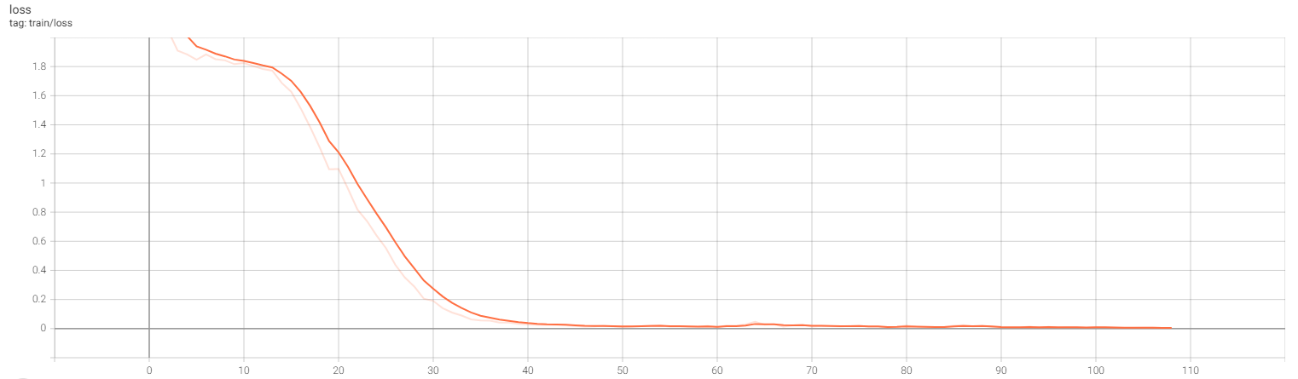


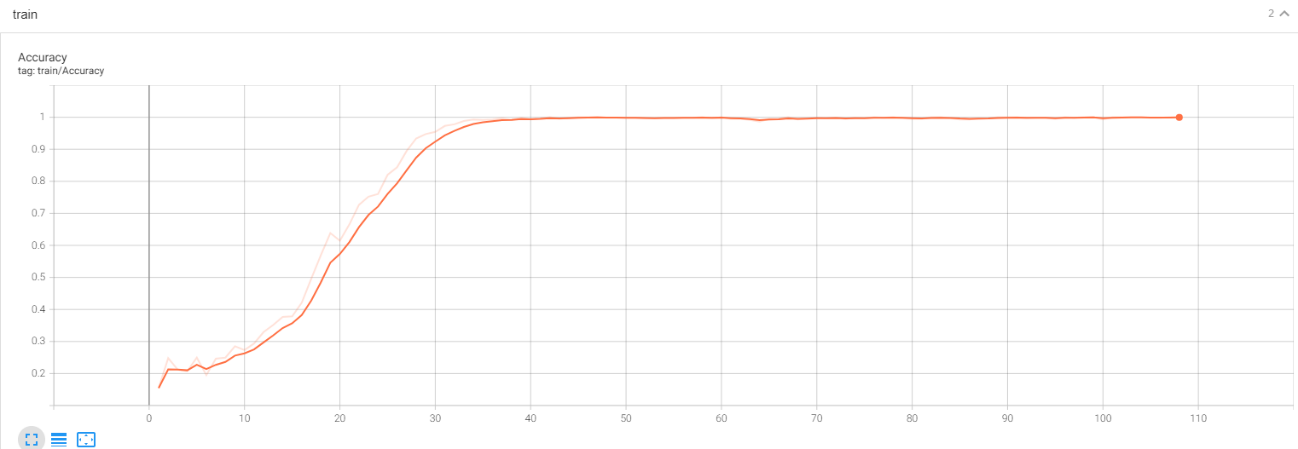


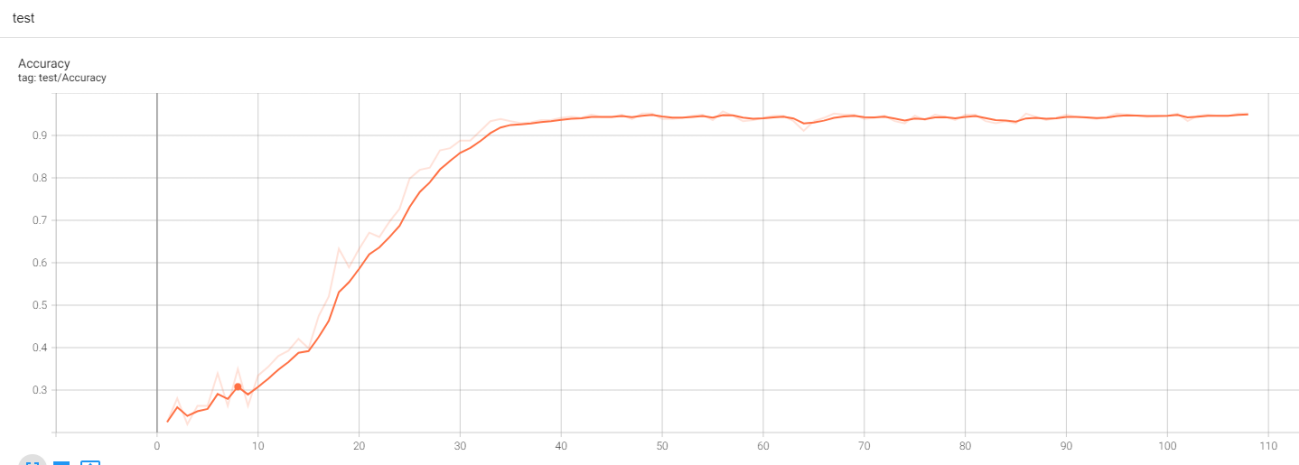


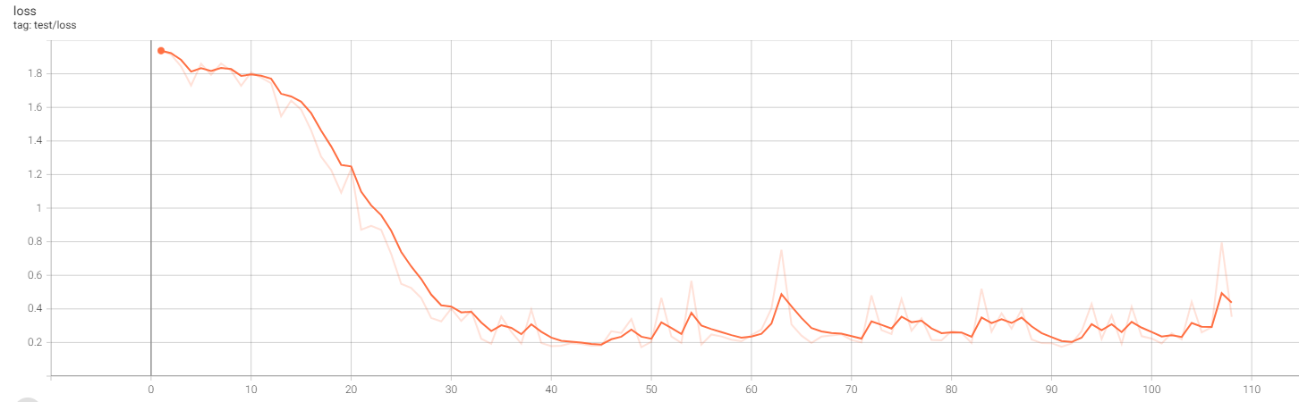


使用AdamW训练108轮(95%准确率)，从上到下分别是训练损失，训练精度，测试精度，测试损失：









**3.4 项目结果展示:**









四、总结

在论文的模型构建过程中，初步学习和了解到一些计算机视觉领域的一些前沿技术与模型，比如本次论文所使用的Swin Transformer模型，它的作者曾经介绍到他主要的思路来源或者说动机是希望可以有一种模型统一NLP和CV领域，这些动机促使他们会去关注一些NLP领域的前沿技术，比如注意力机制和Transformer机制，然后去思考如何将这些技术应用的CV领域，以及NLP模型特征提取和CV模型提取特征的异同，如何对他们进行统一或者说改进，使得这些特征提取方式能够适用于图像特征提取；另一方面是说作者参考了之前ViT的做法，ViT直接将图像切成一个个token送入Transformer网络，它的每一层采样都是一样的，于是作者思考能否将CNN的层次化结构引进，CNN的层次化结构可以使得CNN网络在不同层次关注不同深度的信息，因此层次化结构对网络性能提升起到一定的重要作用，这也是重要的思路来源。

这些作者研究问题的思想来源对我们以后的研究提供给了一些思路，就是说不应该只关注自己所在的研究领域，也应该去关注一些其他相似领域的一些研究成果，往往这些其他相似领域的研究成果会对自己有一些启发，拓展自己的视野与思路，也许困扰很久的问题，就能在其他相似的领域看到解决方案。

五、课程感想与建议

本学期的课程主要学习了深度学习中的一些常见模型，主要包含MLP、CNN和RNN以及RNN的变种LSTM和GRU模型。

在课程的开始，主要对深度学习的基本原理与框架做了一些介绍，深度学习属于联结主义学派，主要思想是利用计算机去模拟人类大脑的神经系统。首先人脑的神经系统是由大量的神经元、树突和轴突组成，在深度学习中通常使用一个加法求和器来代表一个神经元，而树突和轴突则体现在不同神经元之间的权重上；在人脑神经系统中，信息随着电信号转播，而只有信号达到一定的阈值才能通过突触将兴奋信号传递给相邻的神经元，若低于某个阈值则抑制相邻的神经元，深度学习通过使用激活函数来模拟这个过程，同时激活函数还能增加网络的非线性，使得网络有更强的拟合能力，而深度学习网络的训练过程是不断调整权重，这与人类记忆的过程是改变不同神经元之间的连接强度的过程类似。虽然深度学习最初的目的是去模拟人脑神经系统，但随着深度学习的不断发展，这种目的已经逐渐淡化，人们不怎么去关注这个学习过程是否可以在人脑中找到对应的过程，而重点在于能否提升特定任务的识别准确度或者模型能否运行的更快，这种现象也与深度学习模型的不可解释性息息相关，研究可解释的深度学习也是目前的主流研究方向。最后介绍了一些是深度学习网络的基本框架，分为输入层，隐藏层和输出层，隐藏层主要对数据进行特征学习和特征提取，在输出层根据前面隐藏层提取的特征，学习出给的的输出结果格式。

MLP是深度学习中最简单、最基础的模型，其本质是很多个线性模型通过激活函数叠加起来，使其具有非线性能力，从而可以达到很好的学习能力。MLP模型基本有若干个Linear层和激活函数组成，随着网络深度的加深，模型的学习能力也会加强，但是也会带来一些问题，就是梯度消失和过拟合的问题，另一方面也需要更多的训练资源。

虽然MLP网络在一些任务上表现的十分漂亮，但是依然存在很多问题，比如MLP网络由于是全连接，所以参数量十分巨大，随着网络层数的加深就会导致梯度消失的现象出现，而且MLP对于图像问题并不友好，MLP会将图像进行展平，这会导致丧失像素之间的位置信息，因此诞生了CNN网络，CNN受到猫的的大脑皮层信号的启发；具体来说为了解决MLP的问题，CNN具有三大特性，权重共享，局部连接和一定的平移不变性，前两者使得CNN的参数量相比于MLP下降了很多，后者使得CNN对位置不敏感，这一点有利于图像处理问题，因此CNN在图像处理领域有这独特的优势，通过堆叠不同大小卷积核的CNN，可以构建出一种层次化的结构信息，前面提取浅层特征，后面提取深层特征。

虽然CNN在图像领域大放光彩，但是在一些时序性领域却没有那么好的表现，这是因为CNN网络没有记忆性，因此就有了循环神经网络，循环神经网络的主要目的就是给网络增加记忆力，RNN引入所谓的隐状态来表示对过去的记忆，h\_t表面t时刻以前的所有信息，当前时刻的输出取决于h\_t和输入x\_t，并有这二者来计算下一个隐藏状态；RNN的一个缺点是当前记住的信息太多，表示从开始到现在，这很容易导致梯度爆炸或者梯度消失，不容易训练，因此出现了含有遗忘机制的LSTM网络，在该网络中增加了选择门和遗忘门，选择门用来选择需要记住那些信息，遗忘门用来觉得那些信息应该被忘记，为了减少训练参数这些门会共享参数。即便LSTM已经共享门的参数，但是它的门状态依然很多，因此GRU改进了这个问题，GRU相比于LSTM拥少的参数量，因此训练效果也会比LSTM好很多。顺带一提，其实在pytorch中，无论是RNN，LSTM还是GRU，都有高度继承的类，NLP处理的难度在于对文本的预处理，涉及到大量文本操作，预处理难度远高于图像类数据。且在近几年的发展用，循环神经网络逐渐退出历史舞台，原因在于其无法并行化，计算当前状态需要先计算过去的所有状态，因此出现了注意力机制和Transformer机制，这个模型天生容易并行化。

总体来说，本学习的收获还是很多的，对深度学习的各类模型有了一个大体的认识，能够用所学知识去做一些小小的应用。

建议：希望在以后的教学安排中，在CNN之后安排一些CNN的应用，比如YOLO等语义分割或者物体检测算法，虽然难度大了一些，但是这些有趣的项目可以增加学生的兴趣，而不是仅仅局限于屏幕上的训练曲线，通过学习项目，还可以提高代码编写水平，毕竟作为计控的学生，代码能力应该放在首位。

参考文献

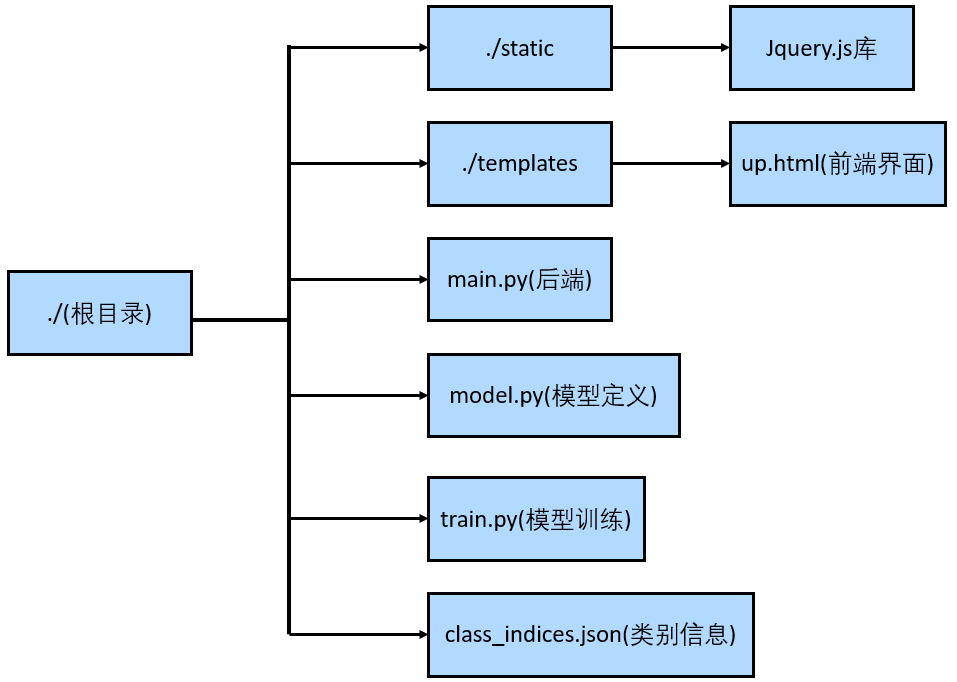
https://www.bilibili.com/video/BV1pL4y1v7jC

<https://www.bilibili.com/video/BV1yg411K7Yc>

https://readpaper.com/paper/3138516171

**附录**

项目目录结构图：



1. up.html：

<!DOCTYPE html>

<html>

<head>

<title>人脸表情识别</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script src="{{ url\_for('static', filename='js/jquery.min.js') }}"></script>

</head>

<body>

<!--<h3>请选择图片文件：PNG/JPG/JPEG/SVG/GIF</h3>-->

<div style="text-align: left;margin-left:500px;margin-top:100px;" >

<h1>人脸表情识别系统</h1>>

<div style="float:left;">

<a href="javascript:;" class="file">选择文件

<input type="file" name="file" id="file0"><br>

</a>

<img src="" id="img0" style="margin-top:20px;width: 35rem;height: 30rem;">

</div>

<div style="float:left;margin-left:50px;">

<input type="button" id="b0" onclick="test()" value="预测">

<pre id="out" style="width:320px;height:50px;line-height: 50px;margin-top:20px;"></pre>

</div>

</div>

<script type="text/javascript">

$("#file0").change(function(){

var objUrl = getObjectURL(this.files[0]) ;//获取文件信息

console.log("objUrl = "+objUrl);

if (objUrl) {

$("#img0").attr("src", objUrl);

}

});

function test() {

var fileobj = $("#file0")[0].files[0];

console.log(fileobj);

var form = new FormData();

form.append("file", fileobj);

var out='';

var flower='';

$.ajax({

type: 'POST',

url: "predict",

data: form,

async: false, //同步执行

processData: false, // 告诉jquery要传输data对象

contentType: false, //告诉jquery不需要增加请求头对于contentType的设置

success: function (arg) {

console.log(arg)

out = arg.result;

},error:function(){

console.log("后台处理错误");

}

});

out.forEach(e=>{

flower+=`<div style="border-bottom: 1px solid #CCCCCC;line-height: 60px;font-size:16px;">${e}</div>`

});

document.getElementById("out").innerHTML=flower;

}

function getObjectURL(file) {

var url = null;

if(window.createObjectURL!=undefined) {

url = window.createObjectURL(file) ;

}else if (window.URL!=undefined) { // mozilla(firefox)

url = window.URL.createObjectURL(file) ;

}else if (window.webkitURL!=undefined) { // webkit or chrome

url = window.webkitURL.createObjectURL(file) ;

}

return url ;

}

</script>

<style>

.file {

position: relative;

/\*display: inline-block;\*/

background: #CCC ;

border: 1px solid #CCC;

padding: 4px 4px;

overflow: hidden;

text-decoration: none;

text-indent: 0;

width:100px;

height:30px;

line-height: 30px;

border-radius: 5px;

color: #333;

font-size: 13px;

}

.file input {

position: absolute;

font-size: 13px;

right: 0;

top: 0;

opacity: 0;

border: 1px solid #333;

padding: 4px 4px;

overflow: hidden;

text-indent: 0;

width:100px;

height:30px;

line-height: 30px;

border-radius: 5px;

color: #FFFFFF;

}

#b0{

background: #1899FF;

border: 1px solid #CCC;

padding: 4px 10px;

overflow: hidden;

text-indent: 0;

width:60px;

height:28px;

line-height: 20px;

border-radius: 5px;

color: #FFFFFF;

font-size: 13px;

}

</style>

</body>

</html>

1. main.py

import os

import io

import json

import torch

import torchvision.transforms as transforms

from PIL import Image

from flask import Flask, jsonify, request, render\_template

from flask\_cors import CORS

from model import SwinTransformer

app = Flask(\_\_name\_\_)

CORS(app) # 解决跨域问题

weights\_path = "./model1.pth"

class\_json\_path = "./class\_indices.json"

assert os.path.exists(weights\_path), "weights path does not exist..."

assert os.path.exists(class\_json\_path), "class json path does not exist..."

# select device

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

print(device)

# create model

model = SwinTransformer(in\_chans=3,

patch\_size=2,

window\_size=5,

embed\_dim=96,

depths=(2, 2, 6, 2),

num\_heads=(3, 6, 12, 24),

num\_classes=7)

# load model weights

model.load\_state\_dict({k.replace('module.',''):v for k,v in torch.load(weights\_path).items()})

model.to(device)

model.eval()

# load class info

json\_file = open(class\_json\_path, 'rb')

class\_indict = json.load(json\_file)

def transform\_image(image\_bytes):

my\_transforms = transforms.Compose([transforms.Resize(50),

transforms.CenterCrop(48),

transforms.ToTensor(),

transforms.Normalize(

[0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])])

image = Image.open(io.BytesIO(image\_bytes)).convert('L').convert('RGB')

if image.mode != "RGB":

raise ValueError("input file does not RGB image...")

return my\_transforms(image).unsqueeze(0).to(device)

def get\_prediction(image\_bytes):

try:

tensor = transform\_image(image\_bytes=image\_bytes)

outputs = torch.softmax(model.forward(tensor).squeeze(), dim=0)

prediction = outputs.detach().cpu().numpy()

template = "class:{:<15} probability:{:.3f}"

index\_pre = [(class\_indict[str(index)], float(p)) for index, p in enumerate(prediction)]

# sort probability

index\_pre.sort(key=lambda x: x[1], reverse=True)

text = [template.format(k, v) for k, v in index\_pre]

return\_info = {"result": text}

except Exception as e:

return\_info = {"result": [str(e)]}

return return\_info

@app.route("/predict", methods=["POST"])

@torch.no\_grad()

def predict():

image = request.files["file"]

img\_bytes = image.read()

info = get\_prediction(image\_bytes=img\_bytes)

return jsonify(info)

@app.route("/", methods=["GET", "POST"])

def root():

return render\_template("up.html")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host="0.0.0.0", port=5000)

1. model.py

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.utils.checkpoint as checkpoint

import numpy as np

from typing import Optional

def drop\_path\_f(x, drop\_prob: float = 0., training: bool = False):

if drop\_prob == 0. or not training:

return x

keep\_prob = 1 - drop\_prob

shape = (x.shape[0],) + (1,) \* (x.ndim - 1) # work with diff dim tensors, not just 2D ConvNets

random\_tensor = keep\_prob + torch.rand(shape, dtype=x.dtype, device=x.device)

random\_tensor.floor\_() # binarize

output = x.div(keep\_prob) \* random\_tensor

return output

class DropPath(nn.Module):

"""Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).

"""

def \_\_init\_\_(self, drop\_prob=None):

super(DropPath, self).\_\_init\_\_()

self.drop\_prob = drop\_prob

def forward(self, x):

return drop\_path\_f(x, self.drop\_prob, self.training)

def window\_partition(x, window\_size: int):

"""

将feature map按照window\_size划分成一个个没有重叠的window

Args:

x: (B, H, W, C)

window\_size (int): window size(M)

Returns:

windows: (num\_windows\*B, window\_size, window\_size, C)

"""

B, H, W, C = x.shape

x = x.view(B, H // window\_size, window\_size, W // window\_size, window\_size, C)

# permute: [B, H//M, M, W//M, M, C] -> [B, H//M, W//M, M, M, C]

# view: [B, H//M, W//M, M, M, C] -> [B\*num\_windows, M, M, C]

windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window\_size, window\_size, C)

return windows

def window\_reverse(windows, window\_size: int, H: int, W: int):

"""

将一个个window还原成一个feature map

Args:

windows: (num\_windows\*B, window\_size, window\_size, C)

window\_size (int): Window size(M)

H (int): Height of image

W (int): Width of image

Returns:

x: (B, H, W, C)

"""

B = int(windows.shape[0] / (H \* W / window\_size / window\_size))

# view: [B\*num\_windows, M, M, C] -> [B, H//M, W//M, M, M, C]

x = windows.view(B, H // window\_size, W // window\_size, window\_size, window\_size, -1)

# permute: [B, H//M, W//M, M, M, C] -> [B, H//M, M, W//M, M, C]

# view: [B, H//M, M, W//M, M, C] -> [B, H, W, C]

x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)

return x

class PatchEmbed(nn.Module):

"""

2D Image to Patch Embedding

"""

def \_\_init\_\_(self, patch\_size=4, in\_c=3, embed\_dim=96, norm\_layer=None):

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

patch\_size = (patch\_size, patch\_size)

self.patch\_size = patch\_size

self.in\_chans = in\_c

self.embed\_dim = embed\_dim

self.proj = nn.Conv2d(in\_c, embed\_dim, kernel\_size=patch\_size, stride=patch\_size)

self.norm = norm\_layer(embed\_dim) if norm\_layer else nn.Identity()

def forward(self, x):

\_, \_, H, W = x.shape

# padding

# 如果输入图片的H，W不是patch\_size的整数倍，需要进行padding

pad\_input = (H % self.patch\_size[0] != 0) or (W % self.patch\_size[1] != 0)

if pad\_input:

# to pad the last 3 dimensions,

# (W\_left, W\_right, H\_top,H\_bottom, C\_front, C\_back)

x = F.pad(x, (0, self.patch\_size[1] - W % self.patch\_size[1],

0, self.patch\_size[0] - H % self.patch\_size[0],

0, 0))

# 下采样patch\_size倍

x = self.proj(x)

\_, \_, H, W = x.shape

# flatten: [B, C, H, W] -> [B, C, HW]

# transpose: [B, C, HW] -> [B, HW, C]

x = x.flatten(2).transpose(1, 2)

x = self.norm(x)

return x, H, W

class PatchMerging(nn.Module):

r""" Patch Merging Layer.

Args:

dim (int): Number of input channels.

norm\_layer (nn.Module, optional): Normalization layer. Default: nn.LayerNorm

"""

def \_\_init\_\_(self, dim, norm\_layer=nn.LayerNorm):

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

self.dim = dim

self.reduction = nn.Linear(4 \* dim, 2 \* dim, bias=False)

self.norm = norm\_layer(4 \* dim)

def forward(self, x, H, W):

"""

x: B, H\*W, C

"""

B, L, C = x.shape

assert L == H \* W, "input feature has wrong size"

x = x.view(B, H, W, C)

# padding

# 如果输入feature map的H，W不是2的整数倍，需要进行padding

pad\_input = (H % 2 == 1) or (W % 2 == 1)

if pad\_input:

# to pad the last 3 dimensions, starting from the last dimension and moving forward.

# (C\_front, C\_back, W\_left, W\_right, H\_top, H\_bottom)

# 注意这里的Tensor通道是[B, H, W, C]，所以会和官方文档有些不同

x = F.pad(x, (0, 0, 0, W % 2, 0, H % 2))

x0 = x[:, 0::2, 0::2, :] # [B, H/2, W/2, C]

x1 = x[:, 1::2, 0::2, :] # [B, H/2, W/2, C]

x2 = x[:, 0::2, 1::2, :] # [B, H/2, W/2, C]

x3 = x[:, 1::2, 1::2, :] # [B, H/2, W/2, C]

x = torch.cat([x0, x1, x2, x3], -1) # [B, H/2, W/2, 4\*C]

x = x.view(B, -1, 4 \* C) # [B, H/2\*W/2, 4\*C]

x = self.norm(x)

x = self.reduction(x) # [B, H/2\*W/2, 2\*C]

return x

class Mlp(nn.Module):

""" MLP as used in Vision Transformer, MLP-Mixer and related networks

"""

def \_\_init\_\_(self, in\_features, hidden\_features=None, out\_features=None, act\_layer=nn.GELU, drop=0.):

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

out\_features = out\_features or in\_features

hidden\_features = hidden\_features or in\_features

self.fc1 = nn.Linear(in\_features, hidden\_features)

self.act = act\_layer()

self.drop1 = nn.Dropout(drop)

self.fc2 = nn.Linear(hidden\_features, out\_features)

self.drop2 = nn.Dropout(drop)

def forward(self, x):

x = self.fc1(x)

x = self.act(x)

x = self.drop1(x)

x = self.fc2(x)

x = self.drop2(x)

return x

class WindowAttention(nn.Module):

def \_\_init\_\_(self, dim, window\_size, num\_heads, qkv\_bias=True, attn\_drop=0., proj\_drop=0.):

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

self.dim = dim

self.window\_size = window\_size # [Mh, Mw]

self.num\_heads = num\_heads

head\_dim = dim // num\_heads

self.scale = head\_dim \*\* -0.5

# define a parameter table of relative position bias

self.relative\_position\_bias\_table = nn.Parameter(

torch.zeros((2 \* window\_size[0] - 1) \* (2 \* window\_size[1] - 1), num\_heads)) # [2\*Mh-1 \* 2\*Mw-1, nH]

# get pair-wise relative position index for each token inside the window

coords\_h = torch.arange(self.window\_size[0])

coords\_w = torch.arange(self.window\_size[1])

coords = torch.stack(torch.meshgrid([coords\_h, coords\_w])) # [2, Mh, Mw]

coords\_flatten = torch.flatten(coords, 1) # [2, Mh\*Mw]

relative\_coords = coords\_flatten[:, :, None] - coords\_flatten[:, None, :] # [2, Mh\*Mw, Mh\*Mw]

relative\_coords = relative\_coords.permute(1, 2, 0).contiguous() # [Mh\*Mw, Mh\*Mw, 2]

relative\_coords[:, :, 0] += self.window\_size[0] - 1 # shift to start from 0

relative\_coords[:, :, 1] += self.window\_size[1] - 1

relative\_coords[:, :, 0] \*= 2 \* self.window\_size[1] - 1

relative\_position\_index = relative\_coords.sum(-1) # [Mh\*Mw, Mh\*Mw]

self.register\_buffer("relative\_position\_index", relative\_position\_index)

self.qkv = nn.Linear(dim, dim \* 3, bias=qkv\_bias)

self.attn\_drop = nn.Dropout(attn\_drop)

self.proj = nn.Linear(dim, dim)

self.proj\_drop = nn.Dropout(proj\_drop)

nn.init.trunc\_normal\_(self.relative\_position\_bias\_table, std=.02)

self.softmax = nn.Softmax(dim=-1)

def forward(self, x, mask: Optional[torch.Tensor] = None):

"""

Args:

x: input features with shape of (num\_windows\*B, Mh\*Mw, C)

mask: (0/-inf) mask with shape of (num\_windows, Wh\*Ww, Wh\*Ww) or None

"""

# [batch\_size\*num\_windows, Mh\*Mw, total\_embed\_dim]

B\_, N, C = x.shape

# qkv(): -> [batch\_size\*num\_windows, Mh\*Mw, 3 \* total\_embed\_dim]

# reshape: -> [batch\_size\*num\_windows, Mh\*Mw, 3, num\_heads, embed\_dim\_per\_head]

# permute: -> [3, batch\_size\*num\_windows, num\_heads, Mh\*Mw, embed\_dim\_per\_head]

qkv = self.qkv(x).reshape(B\_, N, 3, self.num\_heads, C // self.num\_heads).permute(2, 0, 3, 1, 4)

# [batch\_size\*num\_windows, num\_heads, Mh\*Mw, embed\_dim\_per\_head]

q, k, v = qkv.unbind(0) # make torchscript happy (cannot use tensor as tuple)

# transpose: -> [batch\_size\*num\_windows, num\_heads, embed\_dim\_per\_head, Mh\*Mw]

# @: multiply -> [batch\_size\*num\_windows, num\_heads, Mh\*Mw, Mh\*Mw]

q = q \* self.scale

attn = (q @ k.transpose(-2, -1))

# view: [Mh\*Mw\*Mh\*Mw,nH] -> [Mh\*Mw,Mh\*Mw,nH]

relative\_position\_bias = self.relative\_position\_bias\_table[self.relative\_position\_index.view(-1)].view(

self.window\_size[0] \* self.window\_size[1], self.window\_size[0] \* self.window\_size[1], -1)

relative\_position\_bias = relative\_position\_bias.permute(2, 0, 1).contiguous() # [nH, Mh\*Mw, Mh\*Mw]

attn = attn + relative\_position\_bias.unsqueeze(0)

if mask is not None:

# mask: [nW, Mh\*Mw, Mh\*Mw]

nW = mask.shape[0] # num\_windows

# attn.view: [batch\_size, num\_windows, num\_heads, Mh\*Mw, Mh\*Mw]

# mask.unsqueeze: [1, nW, 1, Mh\*Mw, Mh\*Mw]

attn = attn.view(B\_ // nW, nW, self.num\_heads, N, N) + mask.unsqueeze(1).unsqueeze(0)

attn = attn.view(-1, self.num\_heads, N, N)

attn = self.softmax(attn)

else:

attn = self.softmax(attn)

attn = self.attn\_drop(attn)

# @: multiply -> [batch\_size\*num\_windows, num\_heads, Mh\*Mw, embed\_dim\_per\_head]

# transpose: -> [batch\_size\*num\_windows, Mh\*Mw, num\_heads, embed\_dim\_per\_head]

# reshape: -> [batch\_size\*num\_windows, Mh\*Mw, total\_embed\_dim]

x = (attn @ v).transpose(1, 2).reshape(B\_, N, C)

x = self.proj(x)

x = self.proj\_drop(x)

return x

class SwinTransformerBlock(nn.Module):

def \_\_init\_\_(self, dim, num\_heads, window\_size=7, shift\_size=0,

mlp\_ratio=4., qkv\_bias=True, drop=0., attn\_drop=0., drop\_path=0.,

act\_layer=nn.GELU, norm\_layer=nn.LayerNorm):

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

self.dim = dim

self.num\_heads = num\_heads

self.window\_size = window\_size

self.shift\_size = shift\_size

self.mlp\_ratio = mlp\_ratio

assert 0 <= self.shift\_size < self.window\_size, "shift\_size must in 0-window\_size"

self.norm1 = norm\_layer(dim)

self.attn = WindowAttention(

dim, window\_size=(self.window\_size, self.window\_size), num\_heads=num\_heads, qkv\_bias=qkv\_bias,

attn\_drop=attn\_drop, proj\_drop=drop)

self.drop\_path = DropPath(drop\_path) if drop\_path > 0. else nn.Identity()

self.norm2 = norm\_layer(dim)

mlp\_hidden\_dim = int(dim \* mlp\_ratio)

self.mlp = Mlp(in\_features=dim, hidden\_features=mlp\_hidden\_dim, act\_layer=act\_layer, drop=drop)

def forward(self, x, attn\_mask):

H, W = self.H, self.W

B, L, C = x.shape

assert L == H \* W, "input feature has wrong size"

shortcut = x

x = self.norm1(x)

x = x.view(B, H, W, C)

# pad feature maps to multiples of window size

# 把feature map给pad到window size的整数倍

pad\_l = pad\_t = 0

pad\_r = (self.window\_size - W % self.window\_size) % self.window\_size

pad\_b = (self.window\_size - H % self.window\_size) % self.window\_size

x = F.pad(x, (0, 0, pad\_l, pad\_r, pad\_t, pad\_b))

\_, Hp, Wp, \_ = x.shape

# cyclic shift

if self.shift\_size > 0:

shifted\_x = torch.roll(x, shifts=(-self.shift\_size, -self.shift\_size), dims=(1, 2))

else:

shifted\_x = x

attn\_mask = None

# partition windows

x\_windows = window\_partition(shifted\_x, self.window\_size) # [nW\*B, M, M, C]

x\_windows = x\_windows.view(-1, self.window\_size \* self.window\_size, C) # [nW\*B, M\*M, C]

# W-MSA/SW-MSA

attn\_windows = self.attn(x\_windows, mask=attn\_mask) # [nW\*B, M\*M, C]

# merge windows

attn\_windows = attn\_windows.view(-1, self.window\_size, self.window\_size, C) # [nW\*B, M, M, C]

shifted\_x = window\_reverse(attn\_windows, self.window\_size, Hp, Wp) # [B, H', W', C]

# reverse cyclic shift

if self.shift\_size > 0:

x = torch.roll(shifted\_x, shifts=(self.shift\_size, self.shift\_size), dims=(1, 2))

else:

x = shifted\_x

if pad\_r > 0 or pad\_b > 0:

# 把前面pad的数据移除掉

x = x[:, :H, :W, :].contiguous()

x = x.view(B, H \* W, C)

# FFN

x = shortcut + self.drop\_path(x)

x = x + self.drop\_path(self.mlp(self.norm2(x)))

return x

class BasicLayer(nn.Module):

def \_\_init\_\_(self, dim, depth, num\_heads, window\_size,

mlp\_ratio=4., qkv\_bias=True, drop=0., attn\_drop=0.,

drop\_path=0., norm\_layer=nn.LayerNorm, downsample=None, use\_checkpoint=False):

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

self.dim = dim

self.depth = depth

self.window\_size = window\_size

self.use\_checkpoint = use\_checkpoint

self.shift\_size = window\_size // 2

# build blocks

self.blocks = nn.ModuleList([

SwinTransformerBlock(

dim=dim,

num\_heads=num\_heads,

window\_size=window\_size,

shift\_size=0 if (i % 2 == 0) else self.shift\_size,

mlp\_ratio=mlp\_ratio,

qkv\_bias=qkv\_bias,

drop=drop,

attn\_drop=attn\_drop,

drop\_path=drop\_path[i] if isinstance(drop\_path, list) else drop\_path,

norm\_layer=norm\_layer)

for i in range(depth)])

# patch merging layer

if downsample is not None:

self.downsample = downsample(dim=dim, norm\_layer=norm\_layer)

else:

self.downsample = None

def create\_mask(self, x, H, W):

# calculate attention mask for SW-MSA

# 保证Hp和Wp是window\_size的整数倍

Hp = int(np.ceil(H / self.window\_size)) \* self.window\_size

Wp = int(np.ceil(W / self.window\_size)) \* self.window\_size

# 拥有和feature map一样的通道排列顺序，方便后续window\_partition

img\_mask = torch.zeros((1, Hp, Wp, 1), device=x.device) # [1, Hp, Wp, 1]

h\_slices = (slice(0, -self.window\_size),

slice(-self.window\_size, -self.shift\_size),

slice(-self.shift\_size, None))

w\_slices = (slice(0, -self.window\_size),

slice(-self.window\_size, -self.shift\_size),

slice(-self.shift\_size, None))

cnt = 0

for h in h\_slices:

for w in w\_slices:

img\_mask[:, h, w, :] = cnt

cnt += 1

mask\_windows = window\_partition(img\_mask, self.window\_size) # [nW, Mh, Mw, 1]

mask\_windows = mask\_windows.view(-1, self.window\_size \* self.window\_size) # [nW, Mh\*Mw]

attn\_mask = mask\_windows.unsqueeze(1) - mask\_windows.unsqueeze(2)

# [nW, Mh\*Mw, Mh\*Mw]

attn\_mask = attn\_mask.masked\_fill(attn\_mask != 0, float(-100.0)).masked\_fill(attn\_mask == 0, float(0.0))

return attn\_mask

def forward(self, x, H, W):

attn\_mask = self.create\_mask(x, H, W) # # [nW, Mh\*Mw, Mh\*Mw]

for blk in self.blocks:

blk.H, blk.W = H, W

if not torch.jit.is\_scripting() and self.use\_checkpoint:

x = checkpoint.checkpoint(blk, x, attn\_mask)

else:

x = blk(x, attn\_mask)

if self.downsample is not None:

x = self.downsample(x, H, W)

H, W = (H + 1) // 2, (W + 1) // 2

return x, H, W

class SwinTransformer(nn.Module):

def \_\_init\_\_(self, patch\_size=4, in\_chans=3, num\_classes=1000,

embed\_dim=96, depths=(2, 2, 6, 2), num\_heads=(3, 6, 12, 24),

window\_size=7, mlp\_ratio=4., qkv\_bias=True,

drop\_rate=0., attn\_drop\_rate=0., drop\_path\_rate=0.1,

norm\_layer=nn.LayerNorm, patch\_norm=True,

use\_checkpoint=False, \*\*kwargs):

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

self.num\_classes = num\_classes

self.num\_layers = len(depths)

self.embed\_dim = embed\_dim

self.patch\_norm = patch\_norm

# stage4输出特征矩阵的channels

self.num\_features = int(embed\_dim \* 2 \*\* (self.num\_layers - 1))

self.mlp\_ratio = mlp\_ratio

# split image into non-overlapping patches

self.patch\_embed = PatchEmbed(

patch\_size=patch\_size, in\_c=in\_chans, embed\_dim=embed\_dim,

norm\_layer=norm\_layer if self.patch\_norm else None)

self.pos\_drop = nn.Dropout(p=drop\_rate)

# stochastic depth

dpr = [x.item() for x in torch.linspace(0, drop\_path\_rate, sum(depths))] # stochastic depth decay rule

# build layers

self.layers = nn.ModuleList()

for i\_layer in range(self.num\_layers):

layers = BasicLayer(dim=int(embed\_dim \* 2 \*\* i\_layer),

depth=depths[i\_layer],

num\_heads=num\_heads[i\_layer],

window\_size=window\_size,

mlp\_ratio=self.mlp\_ratio,

qkv\_bias=qkv\_bias,

drop=drop\_rate,

attn\_drop=attn\_drop\_rate,

drop\_path=dpr[sum(depths[:i\_layer]):sum(depths[:i\_layer + 1])],

norm\_layer=norm\_layer,

downsample=PatchMerging if (i\_layer < self.num\_layers - 1) else None,

use\_checkpoint=use\_checkpoint)

self.layers.append(layers)

self.norm = norm\_layer(self.num\_features)

self.avgpool = nn.AdaptiveAvgPool1d(1)

self.head = nn.Linear(self.num\_features, num\_classes) if num\_classes > 0 else nn.Identity()

self.apply(self.\_init\_weights)

def \_init\_weights(self, m):

if isinstance(m, nn.Linear):

nn.init.trunc\_normal\_(m.weight, std=.02)

if isinstance(m, nn.Linear) and m.bias is not None:

nn.init.constant\_(m.bias, 0)

elif isinstance(m, nn.LayerNorm):

nn.init.constant\_(m.bias, 0)

nn.init.constant\_(m.weight, 1.0)

def forward(self, x):

# x: [B, L, C]

x, H, W = self.patch\_embed(x)

x = self.pos\_drop(x)

for layer in self.layers:

x, H, W = layer(x, H, W)

x = self.norm(x) # [B, L, C]

x = self.avgpool(x.transpose(1, 2)) # [B, C, 1]

x = torch.flatten(x, 1)

x = self.head(x)

return x

1. train.py

from torchvision.datasets import ImageFolder

from torchvision import transforms

from torch.utils.data import random\_split, DataLoader

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from model import SwinTransformer

import torch

from torch import nn

from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()

# 预处理并加载数据

def load\_dataset(filePath, batch\_sz=128, val\_sz=0.4):

trans = transforms.Compose([transforms.ToTensor()])

dataset = ImageFolder(root=filePath,

transform=trans)

data\_len = len(dataset)

val\_len = int(data\_len \* val\_sz)

train\_len = data\_len - val\_len

train\_dataset, val\_dataset = random\_split(dataset=dataset,

lengths=[train\_len, val\_len])

return (DataLoader(train\_dataset, batch\_size=batch\_sz, shuffle=True),

DataLoader(val\_dataset, batch\_size=batch\_sz, shuffle=True),

dataset.class\_to\_idx)

train\_iter, test\_iter, ss = load\_dataset('./CK+48')

net = SwinTransformer(in\_chans=3,

patch\_size=2,

window\_size=5,

embed\_dim=96,

depths=(2, 2, 6, 2),

num\_heads=(3, 6, 12, 24),

num\_classes=7)

# 每一个训练epoch

def train\_epoch(model, train\_loader, optimizer, loss\_fn, epoch):

size = len(train\_loader.dataset)

num\_batches = len(train\_loader)

model.train() # 将模型设置为训练模式

train\_loss, train\_correct = 0, 0

for batch\_idx, (X, y) in enumerate(train\_loader):

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

# 梯度清零， 反向传播，更新网络参数

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 记录损失与正确率

train\_loss += loss.item()

train\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(X), size,

100. \* batch\_idx / num\_batches, loss.item()))

return train\_loss / num\_batches, train\_correct / size

# 测试epoch

def test\_epoch(model, test\_loader, loss\_fn):

size = len(test\_loader.dataset)

num\_batches = len(test\_loader)

model.eval() # 设为评估模式

test\_loss, test\_correct = 0, 0

# 不记录梯度，节省内存

with torch.no\_grad():

for X, y in test\_loader:

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

test\_loss += loss.item()

test\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

test\_loss /= num\_batches

print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, test\_correct, size, 100. \* test\_correct / size))

return test\_loss, test\_correct / size

# 分类问题使用交叉熵作为损失函数

loss\_fn = nn.CrossEntropyLoss()

# 使用随机梯度下降法更新

trainer = torch.optim.AdamW(net.parameters(), lr = 0.0001, weight\_decay=5E-2)

# 使用DP模式训练

net = nn.DataParallel(net)

# 获取训练数据集和测试数据集

# 训练轮数

num\_epochs = 500

# 记录损失和正确率

train\_loss, train\_accuracy = [], []

test\_loss, test\_accuracy = [], []

for epoch in range(1, num\_epochs + 1):

print(f"Epoch {epoch}\n-------------------------------")

a, b = train\_epoch(net, train\_iter, trainer, loss\_fn, epoch)

train\_loss.append(a)

train\_accuracy.append(b)

c, d = test\_epoch(net, test\_iter, loss\_fn)

test\_loss.append(c)

test\_accuracy.append(d)

writer.add\_scalar('train/loss',scalar\_value= a, global\_step=epoch)

writer.add\_scalar('train/Accuracy', scalar\_value=b, global\_step=epoch)

writer.add\_scalar('test/loss', scalar\_value=c, global\_step=epoch)

writer.add\_scalar('test/Accuracy', scalar\_value=d,global\_step=epoch)

# 保存模型参数

torch.save(net.state\_dict(),'./model.pth')

1. class\_indices.json

{

"0": "愤怒",

"1": "鄙视",

"2": "厌恶",

"3": "害怕",

"4": "快乐",

"5": "悲伤",

"6": "惊喜"

}