暨南大學
JINAN UNIVERSITY

本科课程论文

（2021 — 2022 学年第 一 学期）

论文题目：基于 ViT 实现木薯叶病分类

课 程 名 称：数据科学

课 程 类 别：专业必修课

学 生 姓 名：伍光恒

学　　号：2019051306

学　　院：信息科学技术学院

学　　系：

专　　业：智能科学与技术

任 课 教 师：龚文勇

教 师 单 位：信息科学技术学院

2021 年 12 月 12 日

基于 ViT 实现木薯叶病分类

伍光恒
信息科学技术学院
暨南大学
广州，中国
wuguangheng2002@gmail.com

摘要—虽然 Transformer 架构已经成为自然语言处理任务的事实上的标准，在自然语言处理领域取得了巨大的成功，但是它在计算机视觉方面的应用仍然是有限的。最近也有不少学者提出了用于计算机视觉任务的 Transformer 架构，就比如 Vision Transformer（ViT），它完全去掉了经典 CNNs 中的卷积等操作，取而代之的是纯 Transformer 架构。而且，ViT 已经被证明在图像分类、目标检测和语义图像分割等广泛的视觉应用中可以实现极具竞争力的性能。在这篇文章中，我们将尝试采用 ViT 完成对木薯叶病的分类任务，以此来体会 Transformer 架构在计算机视觉领域的真实应用。实验结果表明，ViT 结构在具体的实际应用中能取得很不错的的表现，尽管它可能需要比较大的训练数据。

***Index Terms*—Vision Transformer（ViT），木薯叶，分类任务**

I. Introduction

作为非洲第二大碳水化合物提供源，木薯是小农种植的关键食品安全作物，因为它可以忍受恶劣的环境条件。撒哈拉以南非洲的至少 80%的家庭农场种植了这种淀粉根，但病毒疾病是导致产量底下的主要原因。在数据科学的帮助下，我们可以识别木薯叶常见疾病，以便可以制定具体的方案来进行治疗。现有的疾病检测方法需要农民请求政府资助的农业专家来帮助检查和诊断植物疾病，这是一件耗费大量人力物力的劳动密集型工作。一个额外的挑战是，有效的解决方案必须在显着的条件限制下表现良好，因为非洲农民只能使用具有低带宽的移动摄像机。

以 Transformer 为代表的基于自注意力机制模型架构已经被广泛应用于自然语言处理（NLP）领域。目前主流的方法是先在大型的文本语料库上进行预训练，然后在较小的针对特定任务的数据集上进行微调。得益于 Transformer 的计算效率与可扩展性，我们可以训练超大规模的模型。然而，在计算机视觉领域，基于卷积的结构仍然是占据主导地位的。受到 NLP 领域成功的启发，有许多方法尝试将 CNN 的结构与自注意力结合，有的将卷积操作全部替换等。后者理论上可以实现高效率的工作，但是在现实应用上仍然存在问题。目前计算机视觉领域的常用架构还是经典的 Res-Net 结构。当在没有强正则化的中等大小的数据集上进行训练时，这些改进的模型产生的精度比大小相似的 ResNets 低几个百分点。这个看似令人沮丧的结果实际上是在意料之中：Transformer 缺少一些 CNN 固有的归纳偏置，例如平移不变性和局部性等，因此在数据量不足的情况下进行训练并不能实现很好的表现。但是，如果在较大数据集上进行训练，大规模的数据量可以弥补归纳偏置不足的缺陷。比如 Vit 经过大规模数据预训练后迁移到小规模数据上可以实现优异的效果。

II. Motivation and Related Works

有监督分类问题一般可以使用传统机器学习方法或深度学习学习方法来完成。一般而言，对于图像分类问题，我们通常会基于卷积神经网络（CNN）来实现，这主要是因为卷积操作可以有效提取图像的特征，进而提高分类的精度。鉴于之前笔者并未接触过有关 Transformer 的相关知识，一直都想感受一下神乎其神的跨界新秀 ViT，在这里就借此机会来学习 Transformer 并尝试进行应用。

我们先来回顾一下 Transformer [1]。由于 Transformer 的架构比较复杂，需要比较多的前置知识，所以这里只简单描述一下 Transformer 中的关键部分。首先一个比较重要的部分是自注意力机制的实现，Transformer 采用了 Scaled Dot-Product Attention 简

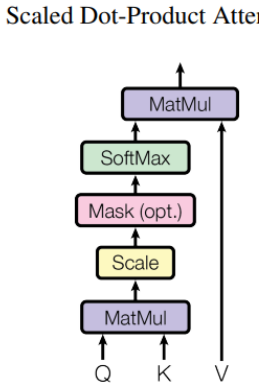

图 1. Scaled Dot-Product Attention

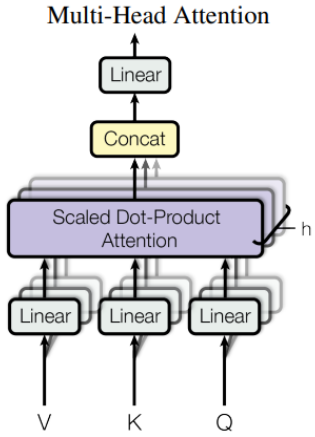

图 2. Multi-head Attention

单地说就是矩阵的运算，不过这里使用了缩放。计算如以下公式

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

所示，缩放的地方在于 $\sqrt{d_k}$。整个过程如图 1所示。然后是 Multi-head Attention 模块。将 V,K,Q 投影至不同的空间，在该空间内完成自注意力的计算，再投影回原来的空间进行拼接，以此来增加表达的可能性。计算可描述如下：

$$MuliHead(Q,K,V) = Concat(head_1,...,head_h)W^O \quad (2)$$

$$head_i = Attention(QW_i^Q, K_i^K, VW_i^V) \quad (3)$$

Multi-head Attention 模块如图 2所示。最后再提一个比较重要的部分 Positional Encoding。Transformer

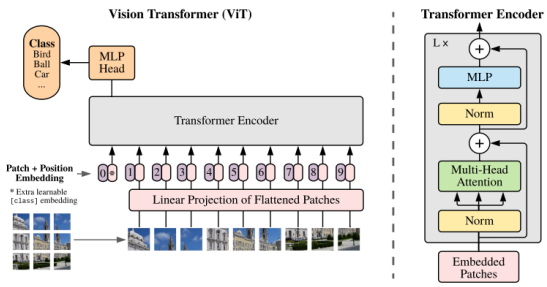

图 3. Model structure of ViT

的注意力是全局注意力，注意力并没有任何关于位置的信息，需要另外加入，Positional Encoding 就是用来完成这一过程。

III. Methods

在这里我们先简要介绍一下 ViT [2]，然后再详细说明我们是如何使用 ViT 的。一般而言，Transformer 接受的输入为序列（token embeddings），为了使其能处理 2D 的图像，在输入之前，ViT 先将图像 reshape 成一系列的展平的 2D 图像块，然后进过线性投影将这些图像块映射到目标维度，得到目标输入向量。然而这些向量并不包含任何和类别相关的信息，要完成图像分类任务，还需另外加入这部分信息。ViT 为图像块嵌入序列预设了一个可学习的嵌入向量——分类头，用于表示图像类别。最终分类时用这个表示类别的向量来进行分类。ViT 比较重要两个部分就如上所述，其他的部分与标准的 Transformer 大同小异，在训练时可能会小的调整，这里不做详细说明。ViT 的结构如图 3所示。

这里我们采用的是预训练的 ViT 模型。主要使用了两个预训练模型：vit_base_patch16_384 和 vit_base_patch16_224 [2]。这里 patch16 表示图像块的大小为 16×16，最后的 384 与 224 表示输入图像的大小，如 384×384。

IV. Experiment

A. Dataset

数据集来源于 Kaggle 上的一个竞赛：Cassava Leaf Disease Classification。数据集总共包含 26337 张图片，总共分以下 5 类：Cassava Bacterial Blight (CBB)，Cassava Brown Streak Disease (CBSD)，Cassava Green Mottle (CGM)，Cassava Mosaic Disease

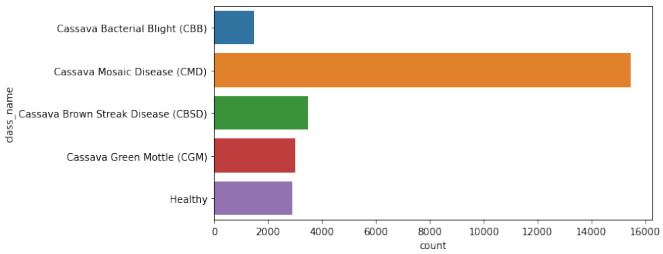

图 4. 类别统计信息

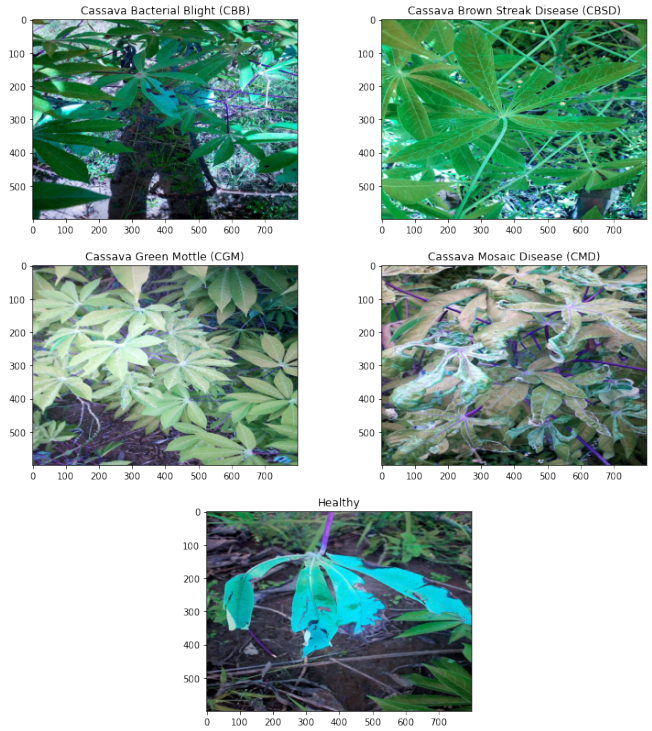

图 5. 5 个类别示例图片

(CMD)，Healthy。各个类别示例图片如图 5所示。类别统计信息如图 4所示。

B. Experimental Settings

首先对输入的图片进行随机裁剪，然后缩放到大小为 384×384，448×448 的图像块。大小为 384×384 的图像块作为模型 vit_base_patch16_384 的输入，大小为 448×448 的图像块进一步被切分成 4 张子图作为模型 vit_base_patch16_224 的输入。而模型 vit_base_patch16_224 中的注意力权重计算又有 A,B 两种模式，具体如图 6所示。所以实质上可认为一共采用了三个模型。其它一些基本的实验设置为：共训练 10 个轮次，批大小设置为 2，损失函数为交叉熵损失函数，

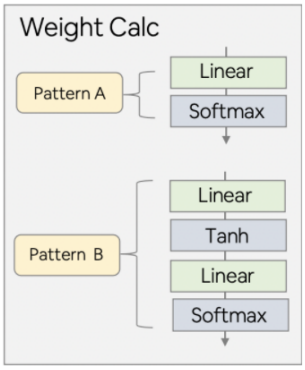

图 6. Attention Calculation Pattern

优化器为 Adam，初始学习率设置为 $1e-4$，采用 5 折交叉验证等。另外还采用了一些辅助训练的手段：

1) Data Augmentation: 为了能更有效地发挥训练集图像的作用，增加鲁棒性，我们对舒颜数据进行了数据增强。采用的数据增强方式有：按一定概率对图像进行水平翻转，垂直翻转，缩放旋转及归一化等操作。数据增强实现如图。

2) LR Scheduler: 对于学习率我们采用了自定义的学习率衰减策略，使得学习率随着迭代次数的增加而减小。调整规则为：

$$lr_{k+1} = lr_k \times \frac{1}{1 + epoch_k} \tag{4}$$

这里 lr_{k+1} 表示下一轮的学习率，lr_k 表示当前学习率，$epoch_k$ 表示当前迭代轮次。

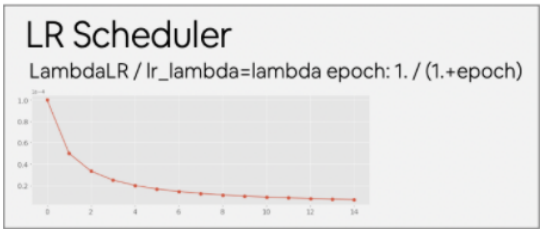

图 7. LR Scheduler

3) Test Time Augmentation(TTA): 测试时数据增强，即在测试时对测试数据进行数据增强，以允许模型对测试数据集中每幅图像的多个不同版本进行预测。对增强图像的预测可以取平均值，从而获得更好的预测性能。通常采用较为简单的数据增强方式，产生较少的副本。本实验测试时数据增强采用的的操作为：将图像 resize 至原来的图像大小及对图像进行归一化。

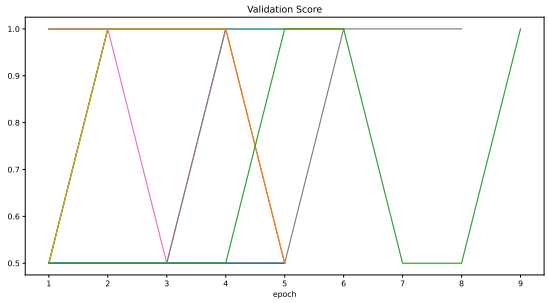

图 8. ViT 模型交叉验证准确率：图中的不同颜色曲线分别表示前面提到的三个模型的 5 折交叉验证结果

C. Experimental Results

由于模型较大，训练时间较长，这里仅选取前 5000 张图像进行训练和验证。最终 5 折交叉验证的准确率为：1.0。这里可能是因为采用数据量较小导致这样的结果。验证的准确率如图 8所示。

由于代码调试花费时间较长，训练过程中经常会出现突然中断现象，花费了大量时间来调试解决这些问题，再加上模型训练参数确实非常多，每进行一次完整的训练都需要耗费大量时间，在实验之前也是没有考虑到实际训练时间如此之长，故许多调整并未来得及实施，现只能给出一个初步的结果，后续可能还需要进一步调整。

V. Conclusion

总的来说，本次大作业尝试应用 ViT 模型完成一个木薯叶病分类任务，但实施起来未尽如人意。首先是代码的调试问题。实验中经常出现的这样的情况：刚开始时代码运行得挺正常，但就会在不知名的地方出错，这种情况很多时候并不是代码本身问题，而是机器的问题，在调试过程中花费了许多时间；第二个是模型的复杂度问题。ViT 模型算是比较复杂的模型，模型计算量较大，刚开始是尝试使用去哪不得数据进行训练，但因为训练时间过长，服务器可能会中断进程导致程序崩溃，所以只能选择较小的数据量来进行训练与测试。但尽管只选取了前 5000 张图像，耗时也长达将近两天花时间。这个是实验之前所没有设想到的。所仅只能仓促出一个结果，来不及进行更多的调整。

通过本次实验，对于数据的处理，程序的设计与实际测试都有了一定的认识。特别是如何应对程序实际运行测试过程中的所可能面对的意料之外的问题。虽然有诸多遗憾，但总归是完成了实验的总体框架，其余的调整实验可在之后进一步完善。

参考文献

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

附录

具体实现代码如下：

Listing 1. **modules.py**

```
import os
import cv2
import math
import random
import numpy as np
import pandas as pd

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset

import timm


class myModel(nn.Module):
    def __init__(self,
                 arch_name,
                 pretrained=False,
                 img_size=256,
                 multi_drop=False,
                 multi_drop_rate=0.5,
                 att_layer=False,
                 att_pattern="A"
                 ):
        super().__init__()
        # 设置可见GPU
        self.device = torch.device(
            "cuda" if torch.cuda.is_available() else "
                cpu")
        self.att_layer = att_layer
        self.multi_drop = multi_drop

        self.model = timm.create_model(
            arch_name, pretrained=pretrained
        )
        n_features = self.model.head.in_features
        self.model.head = nn.Identity()

        self.head = nn.Linear(n_features, 5)
        self.head_drops = nn.ModuleList()
        for i in  range(5):
            self.head_drops.append(nn.Dropout(
                multi_drop_rate))

        if att_layer:
            if att_pattern == "A":
                self.att_layer = nn.Sequential(
                    nn.Linear(n_features, 256),
                    nn.Tanh(),
                    nn.Linear(256, 1),
                )
            elif att_pattern == "B":
                self.att_layer = nn.Linear(n_features,
                    1)
            else:
                raise ValueError("invalid att pattern")

    def forward(self, x):
        if self.att_layer:
            l = x.shape[2] // 2
            h1 = self.model(x[:, :, :l, :l])
            h2 = self.model(x[:, :, :l, l:])
            h3 = self.model(x[:, :, l:, :l])
            h4 = self.model(x[:, :, l:, l:])
            w = F.softmax(torch.cat([
                self.att_layer(h1),
                self.att_layer(h2),
                self.att_layer(h3),
                self.att_layer(h4),
            ], dim=1), dim=1)
            h = h1 * w[:, 0].unsqueeze(-1) + \
                h2 * w[:, 1].unsqueeze(-1) + \
                h3 * w[:, 2].unsqueeze(-1) + \
                h4 * w[:, 3].unsqueeze(-1)
        else:
            h = self.model(x)

        if self.multi_drop:
            for i, dropout in
                enumerate(self.head_drops):
                if i == 0:
                    output = self.head(dropout(h))
                else:
                    output += self.head(dropout(h))
            output /=  len(self.head_drops)
        else:
            output = self.head(h)
        return output
```

```
class myDataset(Dataset):
    def __init__(self,
                 settings,
                 df,
                 transform=None,
                 ):
        self.settings = settings
        self.img_ids = df["image_id"].values
        self.labels = df["label"].values
        self.transform = transform

    def __len__(self):
        return  len(self.img_ids)

    def load_img(self, path):
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        augmented = self.transform(image=image)
        return augmented['image']

    def __getitem__(self, idx):
        img_id = self.img_ids[idx]
        label = torch.tensor(self.labels[idx]). long()
        path = f"{self.settings.DATA_PATH}/train/{
            img_id}"
        return self.load_img(path), label


def linear_combination(x, y, epsilon):
    return epsilon * x + (1 - epsilon) * y


def reduce_loss(loss, reduction='mean'):
    return loss.mean() if reduction == 'mean' else loss
        .
        sum() if reduction == 'sum' else loss


class CrossEntropyLossWithLabelSmooth(nn.Module):  #
    Label smoothing
    def __init__(self, epsilon:
        float = 0.1, reduction='mean'):
         super().__init__()
        self.epsilon = epsilon
        self.reduction = reduction

    def forward(self, preds, target):
        n = preds.size()[-1]
        log_preds = F.log_softmax(preds, dim=-1)
        loss = reduce_loss(-log_preds.
            sum(dim=-1), self.reduction)
        nll = F.nll_loss(log_preds, target, reduction=
            self.reduction)
        return linear_combination(loss / n, nll, self.
            epsilon)
```

Listing 2. **runner.py**

```
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
from collections import OrderedDict

from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold,
    StratifiedKFold

import torch
import torch.nn as nn
from torch.optim import Adam
from torch.nn import CrossEntropyLoss
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import LambdaLR
from albumentations import (
    Compose, Normalize, Resize, RandomResizedCrop,
        HorizontalFlip,
    VerticalFlip, ShiftScaleRotate, Transpose,
)
from albumentations.pytorch import ToTensorV2

from src.utils import get_logger, seed_everything
from src.modules import myModel, myDataset,
    CrossEntropyLossWithLabelSmooth

# 设置可见GPU
os.environ['CUDA_VISIBLE_DEVICES'] = '1,2,3'


class BaseRunner:
    def __init__(self, settings, config):
```

```
        self.settings = settings
        self.config = config
        self.logger = get_logger()

        seed_everything(seed=settings.SEED)

    def split(self, data, target=None, groups=None):
        if self.settings.KFOLD == "StratifiedKFold":
            skf = StratifiedKFold(
                n_splits=self.settings.N_SPLITS,
                    shuffle=True, random_state=self.
                    settings.SEED
            )
            for trn_idx, val_idx in skf.split(data,
                target):
                yield trn_idx, val_idx
        else:
            ValueError(f"invalid settings.KFOLD '{self.
                settings.KFOLD}'")


class Runner(BaseRunner):
    def __init__(self, settings, config):
         super().__init__(settings, config)

        self.device = torch.device(
            "cuda" if torch.cuda.is_available() else "
                cpu")

    def scoring(self, y_true, y_pred):
        return accuracy_score(y_true, y_pred)

    def run(self, is_debug, multi_gpu):
        """ Run Cross-Validation
        """
        df = pd.read_csv(f"{self.settings.DATA_PATH}/
            merged.csv")
        if is_debug:
            df = df.iloc[:500]

        df2020 = df[df["source"] == 2020].iloc[:5000]
            # 只取前10000张图像
        oof = np.zeros((df2020.shape[0], self.settings.
            N_CLASS))
        for fold, (trn_idx, val_idx) in  enumerate(
            self.split(df2020, df2020["label"].values),
                start=1
    ):
        self.logger.info(f"[TRAIN] Fold {fold}")

        # set modules
        self.criterion = CrossEntropyLoss().to(
            self.device
        ) if not self.config.label_smooth else
            CrossEntropyLossWithLabelSmooth(
            epsilon=self.config.label_smooth_alpha
        ).to(self.device)
        self.model = myModel(
            arch_name=self.config.arch_name,
            pretrained=True,
            img_size=self.config.image_size,
            multi_drop=self.config.multi_dropout,
            multi_drop_rate=0.5,
            att_layer=self.config.att_layer,
            att_pattern=self.config.att_pattern,
        )
        if multi_gpu:
            self.model = torch.nn.DataParallel(self
                .model)
        self.model.to(self.device)
        self.optimizer = Adam(self.model.parameters
            (), lr=self.settings.LR)
        self.scheduler = LambdaLR(
            self.optimizer, lr_lambda=lambda epoch:
                1.0 / (1.0 + epoch)
        )

        # split
        train = df2020.iloc[trn_idx] if not self.
            config.use_external else pd.concat([
            df2020.iloc[trn_idx], df[df["source"]
                == 2019]
        ], axis=0)
        valid = df2020.iloc[val_idx]

        # set data_loader
        self.train_loader = DataLoader(
            myDataset(self.settings, train,
                      transform=self.get_transform(
                          is_train=True)),
            # num_workers设为0
            batch_size=self.settings.BATCH_SIZE,
                shuffle=True, drop_last=True,
                num_workers=0,
```

```
        )
        self.valid_loader = DataLoader(
            myDataset(self.settings, valid,
                      transform=self.get_transform(
                          is_train=False)),
            # num_workers设为0
            batch_size=self.settings.BATCH_SIZE,
                shuffle=False, drop_last=False,
                num_workers=0,
        )

        # training
        self.train(fold)

        oof[val_idx, :] = self.predict(self.
            valid_loader)
        fold_score = self.scoring(
            valid["label"].values, oof[val_idx, :].
                argmax(1)
        )
        self.logger.info(f"[RESULT] fold score: {
            fold_score}")

    cv_score = self.scoring(
        df2020["label"].values, oof.argmax(1)
    )
    self.logger.info(f"[RESULT] cv score: {cv_score
        }")
    self.logger.handlers.clear()
    return

def train(self, fold):
    """ Run one fold
    """
    def _train_loop():
        self.model.train()
        total_loss = 0
        for images, labels in tqdm(
            self.train_loader, desc="[TRAIN] train
                loop", leave=False
        ):
            self.optimizer.zero_grad()
            images, labels = images.to(self.device)
                , labels.to(self.device)
            output = self.model(images)
            loss = self.criterion(output, labels)
            loss.backward()
            self.optimizer.step()
            total_loss += loss.detach().cpu()
        total_loss /=  len(self.train_loader)
        return total_loss

    def _valid_loop():
        self.model. eval()
        total_loss = 0
        outputs, targets = [], []
        with torch.no_grad():
            for images, labels in tqdm(
                self.valid_loader, desc="[TRAIN]
                    valid loop", leave=False
            ):
                images, labels = images.to(
                    self.device), labels.to(self.
                        device)
                output = self.model(images)
                output = self.model(images)
                loss = self.criterion(output,
                    labels)
                total_loss += loss.detach().cpu()
                outputs.append(output)
                targets.append(labels)
        total_loss /=  len(self.valid_loader)
        score = self.scoring(
            torch.cat(targets, dim=0).cpu(),
            torch.cat(outputs, dim=0).cpu().argmax
                (1)
        )
        return total_loss, score

    best_eval = 0
    patience = 0
    epoch_trn_loss_values = []
    epoch_val_loss_values = []
    metric_values = []
    model_path = f"{self.settings.OUTPUT_PATH}/{
        self.config.model_name}_{fold}.pth"
    with tqdm(
        range(1, self.settings.EPOCH)) as pbar:
        for epoch in pbar:
            pbar.set_description("[TRAIN] Epoch %d"
                % epoch)
            # train and valid loop
            trn_loss = _train_loop()  # plot
            epoch_trn_loss_values.append(trn_loss)
```

```
            val_loss, score = _valid_loop()  # plot
            epoch_val_loss_values.append(val_loss)
            metric_values.append(score)
            self.scheduler.step()

            if score > best_eval:
                patience, best_eval = 0, score
                torch.save(self.model.state_dict(),
                     model_path)
            else:
                patience += 1
                if patience > self.settings.
                    MAX_PATIENCE:
                    break

            pbar.set_postfix(OrderedDict(
                trn_loss= round(
                    float(trn_loss), 4),
                val_loss= round( float(val_loss),
                    4), val_score=score,
                best_eval=best_eval
            ))

    self.model.load_state_dict(torch.load(
        model_path))
    #plot and save
    self.save_plot(epoch_trn_loss_values, title="
        train")
    self.save_plot(epoch_val_loss_values, title="
        val")
    self.save_plot(metric_values, title="score")

def predict(self, data_loader):
    self.model. eval()
    outputs = []
    func = nn.Softmax(dim=1)
    with torch.no_grad():
        for images, labels in tqdm(data_loader):
            images = images.to(self.device)
            output = self.model(images)
            outputs.append(output)
    return func(torch.cat(outputs, dim=0)).cpu().
        numpy()

def get_transform(self, is_train=True):
    if is_train:
        return Compose([
            RandomResizedCrop(self.config.
                image_size,
                            self.config.
                                image_size),
            Transpose(p=0.5),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.5),
            ShiftScaleRotate(p=0.5),
            Normalize(
                mean=self.settings.MEAN,
                std=self.settings.STD,
            ),
            ToTensorV2(),
        ])
    else:
        return Compose([
            Resize(self.config.image_size, self.
                config.image_size),
            Normalize(
                mean=self.settings.MEAN,
                std=self.settings.STD,
            ),
            ToTensorV2(),
        ])

def save_plot(self, value_list, title):
    plt.figure(title, (12, 6))
    if title == 'score':
        plt.title("Validation Score")
    else:
        plt.title("Epoch Loss")
    x = [i + 1 for i in  range( len(value_list))]
    y = value_list
    plt.xlabel("epoch")
    plt.plot(x, y)
    if title == 'score':
        plt.savefig(os.path.join(
            self.settings.OUTPUT_PATH, 'val_score.
                pdf'))
    else:
        plt.savefig(os.path.join(
            self.settings.OUTPUT_PATH, title+'loss.
                pdf'))
```

Listing 3. **settings.py**

```
""" Base Params
"""
```

```
SEED = 10086
N_CLASS = 5


""" EXP Params
"""
N_SPLITS = 5
KFOLD = "StratifiedKFold"


""" Training Params
"""
LR = 1e-4
EPOCH = 10
MAX_PATIENCE = 3
BATCH_SIZE = 2
MEAN = [0.5, 0.5, 0.5]
STD = [0.5, 0.5, 0.5]


""" PATH (* PATH depends on your env
"""
PATH = "."
DATA_PATH = f"{PATH}/data"
OUTPUT_PATH = f"{PATH}/output"

# ~ - data - train - ***.jpg
#                  |       - ***.jpg ...
#                  - merged.csv
#          - output
```

Listing 4. **utils.py**

```
import os
import torch
import random
import numpy as np
from logging import Formatter, StreamHandler, getLogger


def get_logger():
    log_fmt = Formatter(
        "%(asctime)s [%(levelname)s][%(funcName)s] %(
            message)s "
    )
    logger = getLogger(__name__)
    handler = StreamHandler()
    handler.setLevel("INFO")
    handler.setFormatter(log_fmt)
    logger.setLevel("INFO")
    logger.addHandler(handler)
    logger.propagate = False
    return logger

def seed_everything(seed=1006):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] =  str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True
```

Listing 5. **main.py**

```
import yaml
import pprint
import argparse
from attrdict import AttrDict
from src.runner import Runner
from src import settings


def main(is_debug):
    """ Training Pipeline
    """
    with  open("./config.yaml") as yf:
        config = yaml.safe_load(yf)

    # run single models
    for config_ in config["models"]:
        pprint.pprint(config_)
        runner = Runner(settings, AttrDict(config_))
        runner.run(is_debug=args.debug, multi_gpu=args.
            multi_gpu)

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description="run
        pipeline")
    parser.add_argument("--debug", action="store_true",
         default=False)
    parser.add_argument("--multi-gpu", action="
        store_true", default=False)
    args = parser.parse_args()
```

```
    main(args)
```

Listing 6. **config.yaml**

```yaml
models:
  - arch_name: vit_base_patch16_384
    image_size: 384
    use_external: False
    multi_dropout: False
    att_layer: False
    att_pattern: None
    label_smooth: 0.
    model_name: single_vit
    label_smooth: False
    label_smooth_alpha: 0.

  - arch_name: vit_base_patch16_224
    image_size: 448
    use_external: False
    multi_dropout: False
    att_layer: True
    att_pattern: "A"
    model_name: single_vit4_type_A
    label_smooth: False
    label_smooth_alpha: 0.

  - arch_name: vit_base_patch16_224
    image_size: 448
    use_external: False
    multi_dropout: True
    att_layer: True
    att_pattern: "B"
    model_name: single_vit4_type_B
    label_smooth: True
    label_smooth_alpha: 0.01
```