In [1]:
```
package_path = '../input/pytorch-image-models/pytorch-image-models-master' #'../input/efficientnet-pytorch-07/efficientnet_pytorch-0.7.1'
import sys; sys.path.append(package_path)
```

In [2]:
```
from glob import glob
from sklearn.model_selection import GroupKFold, StratifiedKFold
import cv2
from skimage import io
import torch
from torch import nn
import os
from datetime import datetime
import time
import random
import cv2
import torchvision
from torchvision import transforms
import pandas as pd
import numpy as np
from tqdm import tqdm

import matplotlib.pyplot as plt
from torch.utils.data import Dataset,DataLoader
from torch.utils.data.sampler import SequentialSampler, RandomSampler
from torch.cuda.amp import autocast, GradScaler

import sklearn
import warnings
import joblib
from sklearn.metrics import roc_auc_score, log_loss
from sklearn import metrics
import warnings
import cv2
import pyximport
import timm #from efficientnet_pytorch import EfficientNet
from scipy.ndimage.interpolation import zoom
from sklearn.metrics import log_loss
```

In [3]:
```
CFG = {
    'fold_num': 5,
    'seed': 719,
    'model_arch': 'tf_efficientnet_b4_ns',
    'img_size': 512,
    'epochs': 10,
    'train_bs': 32,
    'valid_bs': 64,
    'lr': 1e-4,
    'num_workers': 4,
    'accum_iter': 1, # suppport to do batch accumulation for backprop with effectively larger batch size
    'verbose_step': 1,
    'device': 'cuda:0',
    'tta': 3,
    'used_epochs': [6,7,8,9],
    'weights': [1,1,1,1]
}
```

In [4]:
```
train = pd.read_csv('../input/cassava-leaf-disease-classification/train.csv')
train.head()
```

Out[4]:

	image_id	label
0	1000015157.jpg	0
1	1000201771.jpg	3
2	100042118.jpg	1
3	1000723321.jpg	1
4	1000812911.jpg	3

In [5]:
```
train.label.value_counts()
```

Out[5]:
```
3    13158
4     2577
2     2386
1     2189
0     1087
Name: label, dtype: int64
```

> We could do stratified validation split in each fold to make each fold's train and validation set looks like the whole train set in target distributions.

In [6]:
```
submission = pd.read_csv('../input/cassava-leaf-disease-classification/sample_submission.csv')
submission.head()
```

Out[6]:

	image_id	label
0	2216849948.jpg	4

Helper Functions

In [7]:
```
def seed_everything(seed):
    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

def get_img(path):
    im_bgr = cv2.imread(path)
    im_rgb = im_bgr[:, :, ::-1]
    #print(im_rgb)
    return im_rgb

img = get_img('../input/cassava-leaf-disease-classification/train_images/1000015157.jpg')
plt.imshow(img)
plt.show()
```

Dataset

In [8]:
```
class CassavaDataset(Dataset):
    def __init__(
        self, df, data_root, transforms=None, output_label=True
    ):

        super().__init__()
        self.df = df.reset_index(drop=True).copy()
        self.transforms = transforms
        self.data_root = data_root
        self.output_label = output_label

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self, index: int):

        # get labels
        if self.output_label:
            target = self.df.iloc[index]["label"]

        path = "{}/{}".format(self.data_root, self.df.iloc[index]['image_id'])

        img  = get_img(path)

        if self.transforms:
            img = self.transforms(image=img)["image"]

        # do label smoothing
        if self.output_label == True:
            return img, target
        else:
            return img
```

Define Train\Validation Image Augmentations

In [9]:
```
from albumentations import *
from albumentations import (
    HorizontalFlip, VerticalFlip, IAAPerspective, ShiftScaleRotate, CLAHE, RandomRotate90,
    Transpose, IAASharpen, IAAEmboss, RandomBrightnessContrast, JpegCompression, Blur,
    IAAPiecewiseAffine, IAAAdditiveGaussianNoise, GaussNoise, MotionBlur, MedianBlur, IAAPiecewiseAffine, RandomResizedCrop
    , IAASharpen, IAAEmboss, RandomBrightnessContrast, Flip, OneOf, Compose, Normalize, Cutout, CoarseDropout, ShiftScaleRotate, CenterCrop, Resize

from albumentations.pytorch import ToTensorV2

from albumentations import *
from albumentations import (
    HorizontalFlip, VerticalFlip, IAAPerspective, ShiftScaleRotate, CLAHE, RandomRotate90,
    Transpose, IAASharpen, IAAEmboss, RandomBrightnessContrast, JpegCompression, Blur,
    IAAPiecewiseAffine, IAAAdditiveGaussianNoise, GaussNoise, MotionBlur, MedianBlur, IAAPiecewiseAffine, RandomResizedCrop
    , IAASharpen, IAAEmboss, RandomBrightnessContrast, Flip, OneOf, Compose, Normalize, Cutout, CoarseDropout, ShiftScaleRotate, CenterCrop, Resize

from albumentations.pytorch import ToTensorV2

def get_train_transforms():
    return Compose([
            RandomResizedCrop(CFG['img_size'], CFG['img_size']),
            Transpose(p=0.5),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.5),
            ShiftScaleRotate(p=0.5),
            HueSaturationValue(hue_shift_limit=0.2, sat_shift_limit=0.2, val_shift_limit=0.2, p=0.5),
            RandomBrightnessContrast(brightness_limit=(-0.1,0.1), contrast_limit=(-0.1, 0.1), p=0.5),
            Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], max_pixel_value=255.0, p=1.0),
            CoarseDropout(p=0.5),
            Cutout(p=0.5),
            ToTensorV2(p=1.0),
        ], p=1.)

def get_valid_transforms():
    return Compose([
            CenterCrop(CFG['img_size'], CFG['img_size'], p=1.),
            Resize(CFG['img_size'], CFG['img_size']),
            Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], max_pixel_value=255.0, p=1.0),
            ToTensorV2(p=1.0),
        ], p=1.)

def get_inference_transforms():
    return Compose([
            RandomResizedCrop(CFG['img_size'], CFG['img_size']),
            Transpose(p=0.5),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.5),
            HueSaturationValue(hue_shift_limit=0.2, sat_shift_limit=0.2, val_shift_limit=0.2, p=0.5),
            RandomBrightnessContrast(brightness_limit=(-0.1,0.1), contrast_limit=(-0.1, 0.1), p=0.5),
            Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], max_pixel_value=255.0, p=1.0),
            ToTensorV2(p=1.0),
        ], p=1.)
```

Model

In [10]:
```
class CassavaImgClassifier(nn.Module):
    def __init__(self, model_arch, n_class, pretrained=False):
        super().__init__()
        self.model = timm.create_model(model_arch, pretrained=pretrained)
        n_features = self.model.classifier.in_features
        self.model.classifier = nn.Linear(n_features, n_class)

    def forward(self, x):
        x = self.model(x)
        return x
```

Main Loop

In [11]:
```
def inference_one_epoch(model, data_loader, device):
    model.eval()

    image_preds_all = []

    pbar = tqdm(enumerate(data_loader), total=len(data_loader))
    for step, (imgs) in pbar:
        imgs = imgs.to(device).float()

        image_preds = model(imgs)   #output = model(input)
        image_preds_all += [torch.softmax(image_preds, 1).detach().cpu().numpy()]


    image_preds_all = np.concatenate(image_preds_all, axis=0)
    return image_preds_all
```

In [12]:
```
if __name__ == '__main__':
    # for training only, need nightly build pytorch

    seed_everything(CFG['seed'])

    folds = StratifiedKFold(n_splits=CFG['fold_num']).split(np.arange(train.shape[0]), train.label.values)

    for fold, (trn_idx, val_idx) in enumerate(folds):
        # we'll train fold 0 first
        if fold > 0:
            break

        print('Inference fold {} started'.format(fold))

        valid_ = train.loc[val_idx,:].reset_index(drop=True)
        valid_ds = CassavaDataset(valid_, '../input/cassava-leaf-disease-classification/train_images/', transforms=get_inference_transforms(), output_label=False)

        test = pd.DataFrame()
        test['image_id'] = list(os.listdir('../input/cassava-leaf-disease-classification/test_images/'))
        test_ds = CassavaDataset(test, '../input/cassava-leaf-disease-classification/test_images/', transforms=get_inference_transforms(), output_label=False)

        val_loader = torch.utils.data.DataLoader(
            valid_ds,
            batch_size=CFG['valid_bs'],
            num_workers=CFG['num_workers'],
            shuffle=False,
            pin_memory=False,
        )

        tst_loader = torch.utils.data.DataLoader(
            test_ds,
            batch_size=CFG['valid_bs'],
            num_workers=CFG['num_workers'],
            shuffle=False,
            pin_memory=False,
        )

        device = torch.device(CFG['device'])
        model = CassavaImgClassifier(CFG['model_arch'], train.label.nunique()).to(device)

        val_preds = []
        tst_preds = []

        #for epoch in range(CFG['epochs']-3):
        for i, epoch in enumerate(CFG['used_epochs']):
            model.load_state_dict(torch.load('../input/pytorch-efficientnet-baseline-train-amp-aug/{}_fold_{}_{}'.format(CFG['model_arch'], fold, epoch)))

            with torch.no_grad():
                for _ in range(CFG['tta']):
                    val_preds += [CFG['weights'][i]/sum(CFG['weights'])/CFG['tta']*inference_one_epoch(model, val_loader, device)]
                    tst_preds += [CFG['weights'][i]/sum(CFG['weights'])/CFG['tta']*inference_one_epoch(model, tst_loader, device)]

        val_preds = np.mean(val_preds, axis=0)
        tst_preds = np.mean(tst_preds, axis=0)

        print('fold {} validation loss = {:.5f}'.format(fold, log_loss(valid_.label.values, val_preds)))
        print('fold {} validation accuracy = {:.5f}'.format(fold, (valid_.label.values==np.argmax(val_preds, axis=1)).mean()))

        del model
        torch.cuda.empty_cache()
```

```
Inference fold 0 started

100%|██████████| 134/134 [00:41<00:00,  3.32it/s]
100%|██████████| 1/1 [00:00<00:00,  1.55it/s]
100%|██████████| 134/134 [01:36<00:00,  1.38it/s]
100%|██████████| 1/1 [00:00<00:00,  4.78it/s]
100%|██████████| 134/134 [01:37<00:00,  1.38it/s]
100%|██████████| 1/1 [00:00<00:00,  7.41it/s]
100%|██████████| 134/134 [01:35<00:00,  1.40it/s]
100%|██████████| 1/1 [00:00<00:00,  5.76it/s]
100%|██████████| 134/134 [01:33<00:00,  1.43it/s]
100%|██████████| 1/1 [00:00<00:00,  8.10it/s]
100%|██████████| 134/134 [01:31<00:00,  1.46it/s]
100%|██████████| 1/1 [00:00<00:00,  6.41it/s]
100%|██████████| 134/134 [01:32<00:00,  1.45it/s]
100%|██████████| 1/1 [00:00<00:00,  5.15it/s]
100%|██████████| 134/134 [01:37<00:00,  1.37it/s]
100%|██████████| 1/1 [00:00<00:00,  4.62it/s]
100%|██████████| 134/134 [01:36<00:00,  1.39it/s]
100%|██████████| 1/1 [00:00<00:00,  7.59it/s]
100%|██████████| 134/134 [01:35<00:00,  1.40it/s]
100%|██████████| 1/1 [00:00<00:00,  8.43it/s]
100%|██████████| 134/134 [01:35<00:00,  1.39it/s]
100%|██████████| 1/1 [00:00<00:00,  7.31it/s]
100%|██████████| 134/134 [01:39<00:00,  1.34it/s]
100%|██████████| 1/1 [00:00<00:00,  7.79it/s]

fold 0 validation loss = 0.21773
fold 0 validation accuracy = 0.93224
```

In [13]:
```
test['label'] = np.argmax(tst_preds, axis=1)
test.head()
```

Out[13]:

	image_id	label
0	2216849948.jpg	4

In [14]:
```
test.to_csv('submission.csv', index=False)
```

Train part is here: https://www.kaggle.com/khyeh0719/pytorch-efficientnet-baseline-train-amp-aug

In [ ]:
```

```