If you fork it, please give an upvote!

About

- In this notebook, we try to improve the score by ensemble.
- I made datasets private.

Source Kernels

- This notebook was written by refering these great kernels below, so please don't forget to **check and upvote** them.
- [No TTA] Cassava Resnext50_32x4d Inference lb0.903
- Clean_Inference_Kernel_8xTTA_LB902
- Cassava-ensemble-(efnetb3-resnet50)

In [1]:
```
#for efficientnet
BATCH_SIZE = 1
image_size = 512
enet_type = ['tf_efficientnet_b4_ns'] * 5
model_path = ['../input/cassava-models-eff/baseline_cld_fold0_epoch8_tf_efficientnet_b4_ns_512.pth',
              '../input/cassava-models-eff/baseline_cld_fold1_epoch9_tf_efficientnet_b4_ns_512.pth',
              '../input/cassava-models-eff/baseline_cld_fold2_epoch9_tf_efficientnet_b4_ns_512.pth',
              '../input/cassava-models-eff/baseline_cld_fold3_epoch5_tf_efficientnet_b4_ns_512.pth',
              '../input/cassava-models-eff/baseline_cld_fold4_epoch11_tf_efficientnet_b4_ns_512.pth']
```

In [2]:
```
# ====================================================
# Library
# ====================================================
import sys
sys.path.append('../input/pytorch-image-models/pytorch-image-models-master')

import os
import math
import time
import random
import shutil
import albumentations
from pathlib import Path
from contextlib import contextmanager
from collections import defaultdict, Counter

import scipy as sp
from scipy.special import softmax
import numpy as np
import pandas as pd

from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold

from tqdm.auto import tqdm
from functools import partial

import cv2
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam, SGD
import torchvision.models as models
from torch.nn.parameter import Parameter
from torch.utils.data import DataLoader, Dataset
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts, CosineAnnealingLR, ReduceLROnPlateau

import albumentations as A
from albumentations.pytorch import ToTensorV2

import timm

import warnings
warnings.filterwarnings('ignore')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [3]:
```
#Transform for efficientnet
transforms_valid = albumentations.Compose([
    albumentations.CenterCrop(image_size, image_size, p=1),
    albumentations.Resize(image_size, image_size),
    albumentations.Normalize()
])
```

Directory settings

In [4]:
```
# ====================================================
# Directory settings for Resnext
# ====================================================
import os

OUTPUT_DIR = './'
MODEL_DIR = '../input/cassava-models-res/'
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

TRAIN_PATH = '../input/cassava-leaf-disease-classification/train_images'
TEST_PATH = '../input/cassava-leaf-disease-classification/test_images'
```

CFG

In [5]:
```
# ====================================================
# CFG for Resnext
# ====================================================
class CFG:
    debug=False
    num_workers=8
    model_name='resnext50_32x4d'
    size=512
    batch_size=32
    seed=2020
    target_size=5
    target_col='label'
    n_fold=5
    trn_fold=[0, 1, 2, 3, 4]
    inference=True
```

Utils

In [6]:
```
# ====================================================
# Utils for Resnext
# ====================================================
def get_score(y_true, y_pred):
    return accuracy_score(y_true, y_pred)


@contextmanager
def timer(name):
    t0 = time.time()
    LOGGER.info(f'[{name}] start')
    yield
    LOGGER.info(f'[{name}] done in {time.time() - t0:.0f} s.')


def init_logger(log_file=OUTPUT_DIR+'inference.log'):
    from logging import getLogger, INFO, FileHandler,  Formatter,  StreamHandler
    logger = getLogger(__name__)
    logger.setLevel(INFO)
    handler1 = StreamHandler()
    handler1.setFormatter(Formatter("%(message)s"))
    handler2 = FileHandler(filename=log_file)
    handler2.setFormatter(Formatter("%(message)s"))
    logger.addHandler(handler1)
    logger.addHandler(handler2)
    return logger

#LOGGER = init_logger()

def seed_torch(seed=42):
    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

seed_torch(seed=CFG.seed)
```

Data Loading

In [7]:
```
test = pd.read_csv('../input/cassava-leaf-disease-classification/sample_submission.csv')
test['filepath'] = test.image_id.apply(lambda x: os.path.join('../input/cassava-leaf-disease-classifica
tion/test_images', f'{x}'))
#test.head()
```

Dataset

In [8]:
```
# ====================================================
# Dataset for Resnext
# ====================================================
class TestDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.file_names = df['image_id'].values
        self.transform = transform

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

    def __getitem__(self, idx):
        file_name = self.file_names[idx]
        file_path = f'{TEST_PATH}/{file_name}'
        image = cv2.imread(file_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform:
            augmented = self.transform(image=image)
            image = augmented['image']
        return image
```

In [9]:
```
# ====================================================
# Dataset for efficientnet
# ====================================================
class CLDDataset(Dataset):
    def __init__(self, df, mode, transform=None):
        self.df = df.reset_index(drop=True)
        self.mode = mode
        self.transform = transform

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

    def __getitem__(self, index):
        row = self.df.loc[index]
        image = cv2.imread(row.filepath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        if self.transform is not None:
            res = self.transform(image=image)
            image = res['image']

        image = image.astype(np.float32)
        image = image.transpose(2,0,1)
        if self.mode == 'test':
            return torch.tensor(image).float()
        else:
            return torch.tensor(image).float(), torch.tensor(row.label).float()

#test = pd.read_csv('../input/cassava-leaf-disease-classification/sample_submission.csv')
#test_dataset = CLDDataset(test, 'test', transform=transforms_valid)
#test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False,  num_wor
kers=4)
```

In [10]:
```
#for efficientnet
test_dataset_efficient = CLDDataset(test, 'test', transform=transforms_valid)
test_loader_efficient = torch.utils.data.DataLoader(test_dataset_efficient, batch_size=BATCH_SIZE, shuf
fle=False,  num_workers=4)
```

In [11]:
```
# ====================================================
# Transforms for Resnext
# ====================================================
def get_transforms(*, data):
    if data == 'valid':
        return A.Compose([
            A.Resize(CFG.size, CFG.size),
            A.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225],
            ),
            ToTensorV2(),
        ])
```

MODELS

In [12]:
```
# ====================================================
# ResNext Model
# ====================================================
class CustomResNext(nn.Module):
    def __init__(self, model_name='resnext50_32x4d', pretrained=False):
        super().__init__()
        self.model = timm.create_model(model_name, pretrained=pretrained)
        n_features = self.model.fc.in_features
        self.model.fc = nn.Linear(n_features, CFG.target_size)

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

In [13]:
```
# ====================================================
# EfficientNet Model
# ====================================================
class enet_v2(nn.Module):

    def __init__(self, backbone, out_dim, pretrained=False):
        super(enet_v2, self).__init__()
        self.enet = timm.create_model(backbone, pretrained=pretrained)
        in_ch = self.enet.classifier.in_features
        self.myfc = nn.Linear(in_ch, out_dim)
        self.enet.classifier = nn.Identity()

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

Helper functions

In [14]:
```
# ====================================================
# Helper functions for Resnext
# ====================================================
def load_state(model_path):
    model = CustomResNext(CFG.model_name, pretrained=False)
    try:  # single GPU model_file
        model.load_state_dict(torch.load(model_path)['model'], strict=True)
        state_dict = torch.load(model_path)['model']
    except:  # multi GPU model_file
        state_dict = torch.load(model_path)['model']
        state_dict = {k[7:] if k.startswith('module.') else k: state_dict[k] for k in state_dict.keys
()}

    return state_dict


def inference(model, states, test_loader, device):
    model.to(device)
    tk0 = tqdm(enumerate(test_loader), total=len(test_loader))
    probs = []
    for i, (images) in tk0:
        images = images.to(device)
        avg_preds = []
        for state in states:
            model.load_state_dict(state)
            model.eval()
            with torch.no_grad():
                y_preds = model(images)
            avg_preds.append(y_preds.softmax(1).to('cpu').numpy())
        avg_preds = np.mean(avg_preds, axis=0)
        probs.append(avg_preds)
    probs = np.concatenate(probs)
    return probs
```

In [15]:
```
# ====================================================
# Helper functions for efficientnet
# ====================================================
def inference_func(test_loader):
    model.eval()
    bar = tqdm(test_loader)

    LOGITS = []
    PREDS = []

    with torch.no_grad():
        for batch_idx, images in enumerate(bar):
            x = images.to(device)
            logits = model(x)
            LOGITS.append(logits.cpu())
            PREDS += [torch.softmax(logits, 1).detach().cpu()]
        PREDS = torch.cat(PREDS).cpu().numpy()
        LOGITS = torch.cat(LOGITS).cpu().numpy()
    return PREDS

def tta_inference_func(test_loader):
    model.eval()
    bar = tqdm(test_loader)
    PREDS = []
    LOGITS = []

    with torch.no_grad():
        for batch_idx, images in enumerate(bar):
            x = images.to(device)
            x = torch.stack([x,x.flip(-1),x.flip(-2),x.flip(-1,-2),
            x.transpose(-1,-2),x.transpose(-1,-2).flip(-1),
            x.transpose(-1,-2).flip(-2),x.transpose(-1,-2).flip(-1,-2)],0)
            x = x.view(-1, 3, image_size, image_size)
            logits = model(x)
            logits = logits.view(BATCH_SIZE, 8, -1).mean(1)
            PREDS += [torch.softmax(logits, 1).detach().cpu()]
            LOGITS.append(logits.cpu())

        PREDS = torch.cat(PREDS).cpu().numpy()

    return PREDS
```

inference and Submit

In [16]:
```
# ====================================================
# inference
# ====================================================

#for Resnext
model = CustomResNext(CFG.model_name, pretrained=False)
#model = enet_v2(enet_type[i], out_dim=5)
states = [load_state(MODEL_DIR+f'{CFG.model_name}_fold{fold}.pth') for fold in CFG.trn_fold]
test_dataset = TestDataset(test, transform=get_transforms(data='valid'))
test_loader = DataLoader(test_dataset, batch_size=CFG.batch_size, shuffle=False,
                         num_workers=CFG.num_workers, pin_memory=True)
predictions = inference(model, states, test_loader, device)

#for Efficientnet
test_preds = []
for i in range(len(enet_type)):
    model = enet_v2(enet_type[i], out_dim=5)
    model = model.to(device)
    model.load_state_dict(torch.load(model_path[i]))
    test_preds += [tta_inference_func(test_loader_efficient)]

# submission
pred = 0.5*predictions + 0.5*np.mean(test_preds, axis=0)
test['label'] = softmax(pred).argmax(1)
test[['image_id', 'label']].to_csv(OUTPUT_DIR+'submission.csv', index=False)
test.head()
```

Out[16]:

	image_id	label	filepath
0	2216849948.jpg	4	../input/cassava-leaf-disease-classification/t...