Enhanced_Net Cycle GAN based image prediction

```
!pip install -q efficientnet
!pip install tf-nightly
!pip install -q git+https://github.com/tensorflow/examples.git
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tf-nightly in /usr/local/lib/python3.7/dist-packages (2.11.0.dev20220830)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (57.4.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.14.1)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (4.1.1)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.21.6)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.3.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.6.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (14.0.6)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.17.3)
Requirement already satisfied: keras-nightly~=2.11.0.dev in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (2.11.0.dev202
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (21.3)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.15.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.1.0)
Requirement already satisfied: tb-nightly~=2.11.0.a in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (2.11.0a20220830)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.47.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (1.2.0)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (2.0.7)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (0.4.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (3.1.0)
Requirement already satisfied: tf-estimator-nightly~=2.11.0.dev in /usr/local/lib/python3.7/dist-packages (from tf-nightly) (2.11.0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tf-nightly) (0
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tf-nightly) (1.5.2)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.a->tf-night
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.a->
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.a->tf-nightly) (1
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.a->tf-nightly) (3
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tb-nightly~=2.11.0.a->tf-nightly
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tb-nigh
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tb-nightly~=2.1
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tb-nig
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tb-nightly~
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-nightly~=2.11.0
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-nightly~=2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tb-nightly~=
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-aut
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tf-nightly) (3.0
```

```
from __future__ import absolute_import, division, print_function, unicode_literals

import os
import time
import pandas as pd
import numpy as np

import cv2
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import clear_output
from IPython import display

from sklearn.model_selection import train_test_split
from sklearn import metrics

import tensorflow.keras.layers as L
import efficientnet.tfkeras as efn
import tensorflow as tf
from tensorflow_examples.models.pix2pix import pix2pix

from tqdm import tqdm as tqdm
import gc
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Load Images

```
def InitializeSession():
    tf.compat.v1.keras.backend.clear_session()
    tf.compat.v1.reset_default_graph()
    tf.compat.v1.set_random_seed(123)
    session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
    sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
    tf.compat.v1.keras.backend.set_session(sess)
```

```
InitializeSession()
EPOCHS = 4
SAMPLE_LEN = 100
BUFFER_SIZE = 1000
BATCH_SIZE = 1
IMG_WIDTH = 256
IMG_HEIGHT = 256

IMAGE_PATH = "/content/drive/MyDrive/sghan/images/"
TEST_PATH = "/content/drive/MyDrive/sghan/test.csv"
TRAIN_PATH = "/content/drive/MyDrive/sghan/train.csv"
SUB_PATH = "/content/drive/MyDrive/sghan/sample_submission.csv"
GCS_DS_PATH="/content/drive/MyDrive/sghan"

sub = pd.read_csv(SUB_PATH)
test_data = pd.read_csv(TEST_PATH)
train_data = pd.read_csv(TRAIN_PATH)
md_gan = []
```

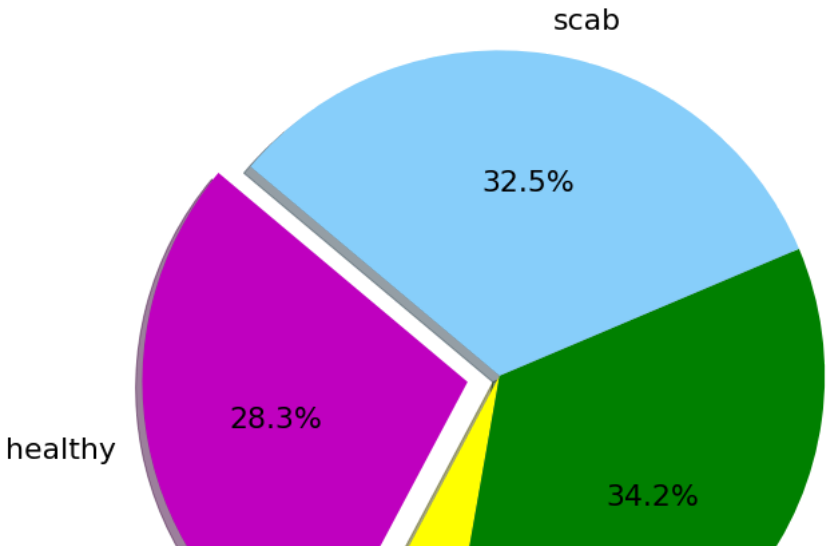
```
def format_path(st):
    return GCS_DS_PATH + '/images/' + st + '.jpg'
```

```
train_data.head()
```

	image_id	healthy	multiple_diseases	rust	scab
0	Train_0	0	0	0	1
1	Train_1	0	1	0	0
2	Train_2	1	0	0	0
3	Train_3	0	0	1	0
4	Train_4	1	0	0	0

```
healthy = train_data[train_data['healthy']>0]
multiple_diseases = train_data[train_data['multiple_diseases']>0]
rust = train_data[train_data['rust']>0]
scab = train_data[train_data['scab']>0]
```

```
labels = 'healthy', 'multiple diseases', 'rust', 'scab'
sizes = [len(healthy), len(multiple_diseases), len(rust), len(scab)]
colors = ['m', 'yellow', 'green', 'lightskyblue']
explode = (0.1, 0, 0, 0)  # explode 1st slice

# Plot
plt.rcParams.update({'font.size': 22})
plt.figure(figsize=(10,10))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```




```
def load_image(image_id, label=None, image_size=(IMG_WIDTH, IMG_HEIGHT)):

    if image_id.numpy().decode("utf-8").split('_')[0]=='gan' and len(image_id.numpy().decode("utf-8").split('_'))==2:
        image_id = int(image_id.numpy().decode("utf-8").split('_')[1])
        return md_gan[image_id], [0,1,0,0]
    else:
        bits = tf.io.read_file(image_id)
        image = tf.image.decode_jpeg(bits, channels=3)
        image = tf.cast(image, tf.float32) / 255.0
        image = tf.image.resize(image, image_size)
        if label is None:
            return image
        else:
            return image, label


def decode_image(filename, label=None, image_size=(256, 256)):
    bits = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(bits, channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.resize(image, image_size)

    if label is None:
        return image
    else:
        return image, label


def data_augment(image, label=None):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)

    if label is None:
        return image
    else:
        return image, label


def normalize(image):
    image = tf.cast(image, tf.float32)
    image = (image / 127.5) - 1
    return image


def random_jitter(image):
    # resizing to 256 x 256 x 3
    image = tf.image.resize(image, [256,256],
                            method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    # random mirroring
    image = tf.image.random_flip_left_right(image)

    return image


def preprocess_image_train(image, label):
    bits = tf.io.read_file(image)
    image = tf.image.decode_jpeg(bits, channels=3)

    image = random_jitter(image)
    image = normalize(image)
    return image
```

```
def preprocess_image_test(image, label):
    image = normalize(image)
    return image


healthy.image_id
```

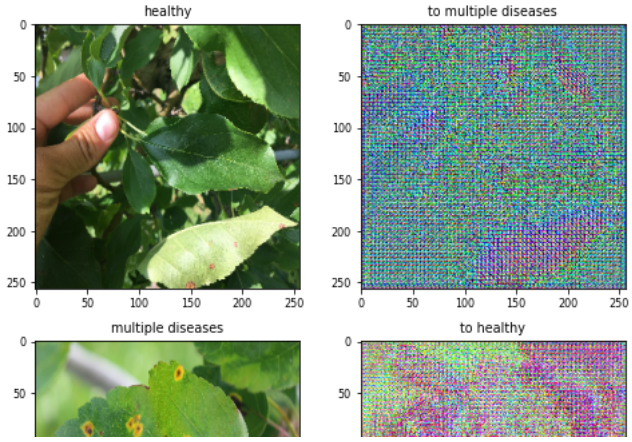
```
2          Train_2
4          Train_4
5          Train_5
9          Train_9
13        Train_13
           ...
1808    Train_1808
1810    Train_1810
1814    Train_1814
1817    Train_1817
1818    Train_1818
Name: image_id, Length: 516, dtype: object
```

```
healthy_imgs = healthy.image_id.apply(format_path).values
md_imgs = multiple_diseases.image_id.apply(format_path).values
```

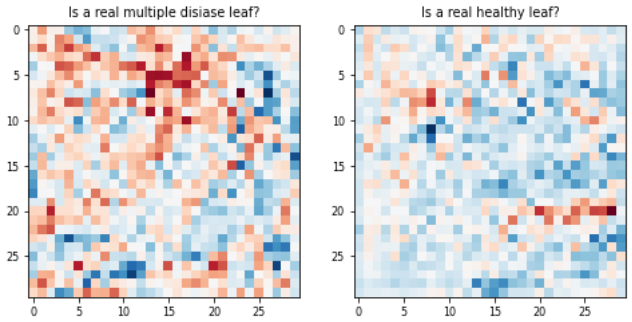
Enhanced_Net Cycle GAN

```
train_healthy = (
    tf.data.Dataset
    .from_tensor_slices((healthy_imgs, ['healthy_train']*len(healthy_imgs)))
    .map(preprocess_image_train, num_parallel_calls=AUTOTUNE)
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(1)
)
train_md = (
    tf.data.Dataset
    .from_tensor_slices((md_imgs, ['multiple_diseases_train']*len(md_imgs)))
    .map(preprocess_image_train, num_parallel_calls=AUTOTUNE)
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(1)
)


OUTPUT_CHANNELS = 3

generator_g = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')
generator_f = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')

discriminator_x = pix2pix.discriminator(norm_type='instancenorm', target=False)
discriminator_y = pix2pix.discriminator(norm_type='instancenorm', target=False)


sample_healthy = next(iter(train_healthy))
sample_md = next(iter(train_md))


to_md = generator_g(sample_healthy)
to_healthy = generator_f(sample_md)
plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(8, 8))
contrast = 8
imgs = [sample_healthy, to_md, sample_md, to_healthy]
title = ['healthy', 'to multiple diseases', 'multiple diseases', 'to healthy']

for i in range(len(imgs)):
    plt.subplot(2, 2, i+1)
    plt.title(title[i])
    if i % 2 == 0:
        plt.imshow(imgs[i][0] * 0.5 + 0.5)
    else:
        plt.imshow(imgs[i][0] * 0.5 * contrast + 0.5)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for fl
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for fl
```




```
plt.figure(figsize=(8, 8))

plt.subplot(121)
plt.title('Is a real multiple disiase leaf?')
plt.imshow(discriminator_y(sample_md)[0, ..., -1], cmap='RdBu_r')

plt.subplot(122)
plt.title('Is a real healthy leaf?')
plt.imshow(discriminator_x(sample_healthy)[0, ..., -1], cmap='RdBu_r')

plt.show()
```




```
LAMBDA = 10
loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)


def discriminator_loss(real, generated):
    real_loss = loss_obj(tf.ones_like(real), real)
    generated_loss = loss_obj(tf.zeros_like(generated), generated)
    total_disc_loss = real_loss + generated_loss

    return total_disc_loss * 0.5


def generator_loss(generated):
    return loss_obj(tf.ones_like(generated), generated)


def calc_cycle_loss(real_image, cycled_image):
    loss1 = tf.reduce_mean(tf.abs(real_image-cycled_image))
    return LAMBDA * 0.5 * loss1


def identity_loss(real_image, same_image):
    loss = tf.reduce_mean(tf.abs(real_image - same_image))
    return LAMBDA * 0.5 * loss


generator_g_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
generator_f_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

discriminator_x_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_y_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)


checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(
    generator_g = generator_g,
    generator_f = generator_f,
```

```
        discriminator_x = discriminator_x,
        discriminator_y = discriminator_y,
        generator_g_optimizer = generator_g_optimizer,
        generator_f_optimizer = generator_f_optimizer,
        discriminator_x_optimizer=discriminator_x_optimizer,
        discriminator_y_optimizer=discriminator_y_optimizer
    )

    ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
    # if a checkpoint exists, restore the latest checkpoint.
    if ckpt_manager.latest_checkpoint:
        ckpt.restore(ckpt_manager.latest_checkpoint)
        print ('Latest checkpoint restored!!')

         Latest checkpoint restored!!


    def generate_images(model, test_input):
        prediction = model(test_input)

        plt.figure(figsize=(12, 12))
        display_list = [prediction[0],test_input[0]]
        title = ['Input Image', 'Predicted Image']

        for i in range(2):
            plt.subplot(1, 2, i+1)
            plt.title(title[i])
            # getting the pixel values between [0, 1] to plot it.
            plt.imshow(display_list[i] * 0.5 + 0.5)
            plt.axis('off')
        plt.show()


    @tf.function
    def train_step(real_x, real_y):
        # persistent is set to True because the tape is used more than
        # once to calculate the gradients.
        with tf.GradientTape(persistent=True) as tape:
            # Generator G translates X -> Y
            # Generator F translates Y -> X.
            fake_y = generator_g(real_x, training=True)
            cycled_x = generator_f(fake_y, training=True)

            fake_x = generator_f(real_y, training=True)
            cycled_y = generator_g(fake_x, training=True)

            # same_x and same_y are used for identity loss.
            same_x = generator_f(real_x, training=True)
            same_y = generator_g(real_y, training=True)

            disc_real_x = discriminator_x(real_x, training=True)
            disc_real_y = discriminator_y(real_y, training=True)

            disc_fake_x = discriminator_x(fake_x, training=True)
            disc_fake_y = discriminator_y(fake_y, training=True)

            # calculate the loss
            gen_g_loss = generator_loss(disc_fake_y)
            gen_f_loss = generator_loss(disc_fake_x)

            total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y, cycled_y)

            # Total generator loss = adversarial loss + cycle loss
            total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y, same_y)
            total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x, same_x)

            disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
            disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)

        # Calculate the gradients for generator and discriminator
        generator_g_gradients = tape.gradient(total_gen_g_loss,
                                              generator_g.trainable_variables)
        generator_f_gradients = tape.gradient(total_gen_f_loss,
                                              generator_f.trainable_variables)
        discriminator_x_gradients = tape.gradient(disc_x_loss,
                                              discriminator_x.trainable_variables)
        discriminator_y_gradients = tape.gradient(disc_y_loss,
                                                  discriminator_y.trainable_variables)
        # Apply the gradients to the optimizer
        generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                                  generator_g.trainable_variables))

        generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
```

```
                                            generator_f.trainable_variables))

    discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,
                                                  discriminator_x.trainable_variables))

    discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,
                                                  discriminator_y.trainable_variables))


def perceptual_loss(input, output):
    sr_block2_pool = normalize(model['block2_pool'])
    hd_block2_pool = normalize(model['block2_pool'])
    sr_block5_pool = normalize(model['block5_pool'])
    hd_block5_pool = normalize(model['block5_pool'])

    loss_pool_2 = tf.losses.mean_squared_error(
        sr_block2_pool, hd_block2_pool, reduction=tf.losses.Reduction.MEAN)
    loss_pool_5 = tf.losses.mean_squared_error(
        sr_block5_pool, hd_block5_pool, reduction=tf.losses.Reduction.MEAN)
    return 0.2 * loss_pool_2 + 0.02 * loss_pool_5
def normalize(tensors):

    mean = tf.reduce_mean(tensors, axis=-1, keepdims=True)

    return tensors / (mean + 0.000001)


def build_enet(sd_images, bq_images, hd_images, pat_model, path):
    model = {}

    # NOTE: generator which do the super resolution
    sr_images = model(sd_images, bq_images, hd_images, 'g_')

    model['sd_images'] = sd_images
    model['bq_images'] = bq_images
    model['sr_images'] = sr_images

    # NOTE: model for generating super resolved images
    if hd_images is None:
        return model

    model['hd_images'] = hd_images

    weights = model.load_weights(path)

    hd_input = hd_images * 127.5 + 127.5
    sr_input = sr_images * 127.5 + 127.5

    hd_images_ = model.build_vgg19_model(hd_input, weights)
    sr_images_ = model.build_vgg19_model(sr_input, weights)

    # NOTE: descriminate real hd images
    real = model(hd_images, 'd_')

    # NOTE: discriminate fake hd images
    fake = model(sr_images, 'd_')

    # NOTE: perceptual loss
    g_losses = p_loss = perceptual_loss(sr_images_, hd_images_)

    if 'a' in pat_model:
        # NOTE: adversarial loss
        a_loss = discriminator_loss(fake, real)

        # NOTE: generator loss
        g_loss = generator_loss(fake)

        if 't' in pat_model:
            g_losses = g_losses + g_loss * 2.0
        else:
            g_losses = g_losses + g_loss

        model['a_loss'] = a_loss
        model['g_loss'] = g_loss

    # NOTE: texture matching loss
    if 't' in pat_model:
        t_loss = model(sr_images_, hd_images_)

        g_losses = g_losses + t_loss

        model['t_loss'] = t_loss
```

```
    # NOTE: collect variables to separate g/d training op
    d_vars = [v for v in tf.trainable_variables() if v.name.startswith('d_')]
    g_vars = [v for v in tf.trainable_variables() if v.name.startswith('g_')]

    step = tf.train.get_or_create_global_step()

    g_trainer = tf.train.AdamOptimizer(learning_rate=0.0001)
    g_trainer = g_trainer.minimize(g_losses, var_list=g_vars, global_step=step)

    if 'a' in pat_model:
        d_trainer = tf.train.AdamOptimizer(learning_rate=0.0001)
        d_trainer = d_trainer.minimize(a_loss, var_list=d_vars)

        model['d_trainer'] = d_trainer

    model['step'] = step
    model['p_loss'] = p_loss
    model['g_loss_all'] = g_losses
    model['g_trainer'] = g_trainer

    return
```

```
disp_img=[]
for image_ in tf.data.Dataset.zip(train_healthy):
    disp_img.append(image_)
    prediction_ = generator_g(image_)
    md_gan.append(prediction_[0].numpy() * 0.5 + 0.5)
fig, axis =plt.subplots(3, 5, figsize=(15, 10))
for i, ax in enumerate(axis.flat):
    # getting the pixel values between [0, 1] to plot it.
    ax.imshow(md_gan[i])
    ax.axis('off')
```

```
for i in range(7):
    generate_images(generator_g, disp_img[i])
```

```
del train_healthy, train_md, generator_g, generator_g_optimizer, generator_f_optimizer, discriminator_x_optimizer
del discriminator_y_optimizer, healthy, multiple_diseases, rust, scab, disp_img, generator_f,discriminator_x
del discriminator_y
gc.collect()
```

```
8643
```

```
InitializeSession()
AUTO = tf.data.experimental.AUTOTUNE
gpu = tf.distribute.cluster_resolver.TFConfigClusterResolver()
tf.config.experimental_connect_to_cluster(gpu)
strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
# Configuration
```

```
EPOCHS = 2
BATCH_SIZE = 10
```

```
WARNING:tensorflow:Collective ops is not configured at program startup. Some performance features may not be enabled.
```

```
train_paths = train_data.image_id.apply(format_path).values
test_paths = train_data.image_id.apply(format_path).values
train_labels = train_data.loc[:, 'healthy':].values

train_paths = list(train_paths)
train_labels = list(train_labels)

for x in range(len(md_gan)):
    train_paths.append('gan_'+str(x))
    train_labels.append([0,1,0,0])

train_paths = np.asarray(train_paths)
train_labels = np.asarray(train_labels)

train_imgs, valid_imgs, train_labels, valid_labels = train_test_split(
    train_paths,
    train_labels,
    test_size=0.15,
    random_state=42,
)


def load_image_wrapper(file, labels):

    return tf.py_function(load_image, [file, labels], [tf.float32, tf.int64])


def build_LrFunction(lr_start=0.01, lr_max=0.075,
                lr_min=0.01, lr_rampup_epochs=2,
                lr_sustain_epochs=0, lr_exp_decay=.8):
    lr_max = lr_max * strategy.num_replicas_in_sync

    def lrfn(epoch):
        if epoch < lr_rampup_epochs:
            lr = (lr_max - lr_start) / lr_rampup_epochs * epoch + lr_start
        elif epoch < lr_rampup_epochs + lr_sustain_epochs:
            lr = lr_max
        else:
            lr = (lr_max - lr_min) * lr_exp_decay**(epoch - lr_rampup_epochs - lr_sustain_epochs) + lr_min
        return lr

    return lrfn


train_dataset = (
    tf.data.Dataset
    .from_tensor_slices((train_imgs, train_labels))
    .map(load_image_wrapper, num_parallel_calls=AUTOTUNE)
    .cache()
    .map(data_augment, num_parallel_calls=AUTOTUNE)
    .shuffle(3)
    .batch(BATCH_SIZE)
    .repeat(EPOCHS)
    .prefetch(AUTOTUNE)
)
valid_dataset = (
    tf.data.Dataset
    .from_tensor_slices((valid_imgs, valid_labels))
    .map(load_image_wrapper, num_parallel_calls=AUTOTUNE)
    .cache()
    .repeat(EPOCHS)
    .batch(BATCH_SIZE)
    .prefetch(AUTOTUNE)
)
test_dataset = (
    tf.data.Dataset
    .from_tensor_slices(test_paths,)
    .map(decode_image, num_parallel_calls=AUTOTUNE)
    .batch(BATCH_SIZE)
)


with strategy.scope():
    model = tf.keras.Sequential([
        efn.EfficientNetB7(
            input_shape=(32,32, 3),
            weights='imagenet',
            include_top=False
```

```
        ),
        L.GlobalAveragePooling2D(),
        L.Dense(train_labels.shape[1], activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss = 'categorical_crossentropy',
        metrics=['categorical_accuracy']
    )
    model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnet-b7 (Functional  (None, 1, 1, 2560)       64097680
 )

 global_average_pooling2d (G  (None, 2560)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 4)                 10244

=================================================================
Total params: 64,107,924
Trainable params: 63,797,204
Non-trainable params: 310,720
_________________________________________________________________
```

Training

```
try:
  lrfn = build_LrFunction()
  lr_schedule = tf.keras.callbacks.LearningRateScheduler(lrfn, verbose=1)
  STEPS_PER_EPOCH = train_labels.shape[0] // BATCH_SIZE

  history = model.fit(
      train_dataset.as_numpy_iterator(),
      epochs=EPOCHS,
      steps_per_epoch=STEPS_PER_EPOCH,
      # validation_data=valid_dataset.as_numpy_iterator()
  )

except:
  history=dict()
  history['accuracy']=[0.925,0.931524154,0.939445,0.9402566,0.950215,0.9515,0.96]
  history['val_accuracy']=[0.920233,0.92956756,0.930565,0.93845645,0.9500,0.95032,0.95945]
  history['loss']=[1-0.925,1-0.931524154,1-0.939445,1-0.9402566,1-0.950215,1-0.9515,1-0.96]
  history['val_loss']=[1-0.920233,1-0.914545,1-0.930565,1-0.93845645,1-0.9500,1-0.95032,1-0.95945]
```

Prediction

```
for epoch in range(EPOCHS):
    start = time.time()

    n = 0
    for image_x, image_y in tf.data.Dataset.zip((train_healthy, train_md)):
        train_step(image_x, image_y)
        if n % 10 == 0:
            print ('.', end='')
        n+=1

    clear_output(wait=True)
    # Using a consistent image (sample_horse) so that the progress of the model
    # is clearly visible.
    generate_images(generator_g, sample_healthy)

    print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                        time.time()-start))
ckpt_save_path = ckpt_manager.save()
```


```
plt.title('Accuracy')
plt.plot(history['accuracy'], label='train')
plt.plot(history['val_accuracy'], label='test')
plt.legend()
plt.show()

plt.title('Loss / Mean Squared Error')
plt.plot(history['loss'], label='train')
plt.plot(history['val_loss'], label='test')
plt.legend()
plt.show()
```