Import

```
 1
 2
 3 import os
 4 import sys
 5 from tempfile import NamedTemporaryFile
 6 from urllib.request import urlopen
 7 from urllib.parse import unquote, urlparse
 8 from urllib.error import HTTPError
 9 from zipfile import ZipFile
10 import tarfile
11 import shutil
12
13 CHUNK_SIZE = 40960
14 DATA_SOURCE_MAPPING = 'horse2zebra-dataset:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F926321%2F1567596%2Ft
15
16 KAGGLE_INPUT_PATH='/kaggle/input'
17 KAGGLE_WORKING_PATH='/kaggle/working'
18 KAGGLE_SYMLINK='kaggle'
19
20 !umount /kaggle/input/ 2> /dev/null
21 shutil.rmtree('/kaggle/input', ignore_errors=True)
22 os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
23 os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)
24
25 try:
26   os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
27 except FileExistsError:
28   pass
29 try:
30   os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
31 except FileExistsError:
32   pass
33
34 for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
35     directory, download_url_encoded = data_source_mapping.split(':')
36     download_url = unquote(download_url_encoded)
37     filename = urlparse(download_url).path
38     destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
39     try:
40         with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
41             total_length = fileres.headers['content-length']
42             print(f'Downloading {directory}, {total_length} bytes compressed')
43             dl = 0
44             data = fileres.read(CHUNK_SIZE)
45             while len(data) > 0:
46                 dl += len(data)
47                 tfile.write(data)
48                 done = int(50 * dl / int(total_length))
49                 sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
50                 sys.stdout.flush()
51                 data = fileres.read(CHUNK_SIZE)
52             if filename.endswith('.zip'):
53               with ZipFile(tfile) as zfile:
54                 zfile.extractall(destination_path)
55             else:
56               with tarfile.open(tfile.name) as tarfile:
57                 tarfile.extractall(destination_path)
58             print(f'\nDownloaded and uncompressed: {directory}')
59     except HTTPError as e:
60         print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
61         continue
62     except OSError as e:
63         print(f'Failed to load {download_url} to path {destination_path}')
64         continue
65
66 print('Data source import complete.')
67
```

```
Downloading horse2zebra-dataset, 116809009 bytes compressed
[==================================================] 116809009 bytes downloaded
Downloaded and uncompressed: horse2zebra-dataset
Downloading imagetoimage-translation-cyclegan, 523920850 bytes compressed
[==================================================] 523920850 bytes downloaded
Downloaded and uncompressed: imagetoimage-translation-cyclegan
Data source import complete.
```

Double-click (or enter) to edit

```
from IPython.display import clear_output as clear
!pip install tensorflow-addons
clear()
```

```
# Common
import os
import keras
import numpy as np
from glob import glob
from tqdm import tqdm
import tensorflow as tf
from random import random

# Data
import tensorflow.image as tfi
import matplotlib.pyplot as plt
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

# Model Layers
from keras.layers import ReLU
from keras.layers import Input
from keras.layers import Conv2D
from keras.layers import Dropout
from keras.layers import LeakyReLU
from keras.layers import Activation
from keras.layers import concatenate
from keras.layers import ZeroPadding2D
from keras.layers import Conv2DTranspose
from tensorflow_addons.layers import InstanceNormalization

# Model Functions
from keras.models import Model
from keras.models import load_model
from keras.models import Sequential
from keras.initializers import RandomNormal

# Optimizers
from tensorflow.keras.optimizers import Adam

# Loss
from keras.losses import BinaryCrossentropy

# Model Viz
from tensorflow.keras.utils import plot_model
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.
Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras,

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
```

Custom Functions

```
def show_image(image, title=None):
    '''
    The function takes in an image and plots it using Matplotlib.
    '''
    plt.imshow(image)
    plt.title(title)
    plt.axis('off')
```

Data

```
root_horse_path = '../input/horse2zebra-dataset/trainA'
root_zebra_path = '../input/horse2zebra-dataset/trainB'
horse_paths = sorted(glob(root_horse_path + '/*.jpg'))[:1000]
zebra_paths = sorted(glob(root_zebra_path + '/*.jpg'))[:1000]
```

```
SIZE = 256
horse_images, zebra_images = np.zeros(shape=(len(horse_paths),SIZE,SIZE,3)), np.zeros(shape=(len(horse_paths),SIZE,SIZE,3
for i,(horse_path, zebra_path) in tqdm(enumerate(zip(horse_paths, zebra_paths)), desc='Loading'):

    horse_image = img_to_array(load_img(horse_path))
    horse_image = tfi.resize(tf.cast(horse_image, tf.float32)/255., (SIZE, SIZE))

    zebra_image = img_to_array(load_img(zebra_path))
    zebra_image = tfi.resize(tf.cast(zebra_image,tf.float32)/255., (SIZE, SIZE))

    # as the data is unpaired so we don't have to worry about, positioning the images.

    horse_images[i] = horse_image
    zebra_images[i] = zebra_image
```

```
Loading: 1000it [00:10, 98.83it/s]
```

```
dataset = [horse_images, zebra_images]
```

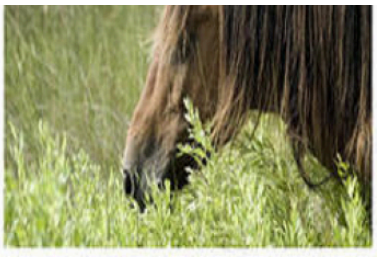
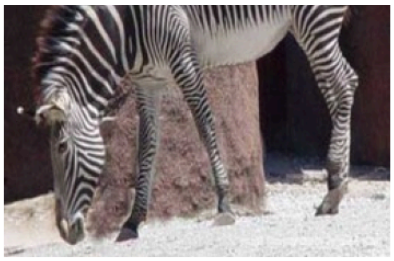
Data Viz

```
# Visualizing
for i in range(10):
    id = np.random.randint(len(horse_images))
    horse, zebra = horse_images[id], zebra_images[id]

    plt.figure(figsize=(10,8))

    plt.subplot(1,2,1)
    show_image(horse)

    plt.subplot(1,2,2)
    show_image(zebra)
    plt.show()
```




© Don Getty

Generator

```
def ResidualBlock(filters, layer, index):
#     init = RandomNormal(stddev=0.02)

    x = Conv2D(filters, kernel_size=3, strides=1, padding='same', kernel_initializer='he_normal', use_bias=False, name="E
    x = InstanceNormalization(axis=-1, name="Block_{}_Normalization1".format(index))(x)
    x = ReLU(name="Block_{}_ReLU".format(index))(x)

    x = Conv2D(filters, kernel_size=3, strides=1, padding='same', kernel_initializer='he_normal', use_bias=False, name="E
    x = InstanceNormalization(axis=-1, name="Block_{}_Normalization2".format(index))(x)

    x = concatenate([x, layer], name="Block_{}_Merge".format(index))

    return x
```

Let's create the **main model architecture**.

```
def downsample(filters, layer, size=3, strides=2, activation=None, index=None, norm=True):
    x = Conv2D(filters, kernel_size=size, strides=strides, padding='same', kernel_initializer='he_normal', use_bias=False
    if norm:
        x = InstanceNormalization(axis=-1, name="Encoder_{}_Normalization".format(index))(x)
    if activation is not None:
        x = Activation(activation, name="Encoder_{}_Activation".format(index))(x)
    else:
        x = LeakyReLU( name="Encoder_{}_LeakyReLU".format(index))(x)
    return x
```

The **downsample layer**, will simply **decrease the size of the input image by 2**.

```
def upsample(filters, layer, size=3, strides=2, index=None):
    x = Conv2DTranspose(filters, kernel_size=size, strides=strides, padding='same', kernel_initializer='he_normal', use_b
    x = InstanceNormalization(axis=-1, name="Decoder_{}_Normalization".format(index))(x)
    x = ReLU( name="Encoder_{}_ReLU".format(index))(x)
    return x
```

Upsample will perform the **exact opposite** and will **increase the image size by 2**.

```
def Generator(n_resnet=9, name="Generator"):

    inp_image = Input(shape=(SIZE, SIZE, 3), name="InputImage")         # 256 x 256 x3

    x = downsample(64, inp_image, size=7, strides=1, index=1)           # 256 x 256 x 64
    x = downsample(128, x, index=2)                                     # 128 x 128 x 128
    x = downsample(256, x, index=3)                                     # 64 x 64 x 256

    for i in range(n_resnet):
        x = ResidualBlock(256, x, index=i+4)                             # (64 x 64 x 256) x n_resnet

    x = upsample(128, x, index=13)                                       # 128 x 128 x 128
    x = upsample(64, x, index=14)                                        # 256 x 256 x 64
    x = downsample(3, x, size=7, strides=1, activation='tanh', index=15) # 256 x 256 x 3

    model = Model(
        inputs=inp_image,
        outputs=x,
        name=name
    )
    return model
```

Discriminator

```
def Discriminator(name='Discriminator'):
    init = RandomNormal(stddev=0.02)
    src_img = Input(shape=(SIZE, SIZE, 3), name="InputImage")     # 256 x 256 x 3
    x = downsample(64, src_img, size=4, strides=2, index=1, norm=False) # 128 x 128 x 64
    x = downsample(128, x, size=4, strides=2, index=2)            # 64 x 64 x 128
    x = downsample(256, x, size=4, strides=2, index=3)            # 32 x 32 x 256
    x = downsample(512, x, size=4, strides=2, index=4)            # 16 x 16 x 512
    x = downsample(512, x, size=4, strides=2, index=5)            # 8 x 8 x 512  # we can try out a different architectur
    patch_out = Conv2D(1, kernel_size=4, padding='same', kernel_initializer=init, use_bias=False)(x) # 8 x 8 x 1

    model = Model(
        inputs=src_img,
        outputs=patch_out,
        name=name
    )
    model.compile(
        loss='mse',
        optimizer=Adam(learning_rate=2e-4, beta_1=0.5),
        loss_weights=[0.5]
    )
    return model
```

Training Functions

```
def CombineModel(g_model1, g_model2, d_model, name):
    # train the Generator
    g_model1.trainable = True

    # Stop the Discriminator and 2nd Generator
    d_model.trainable = False
    g_model2.trainable = False

    # Adversarial Loss
    input_gen = Input(shape=(SIZE, SIZE, 3))
    gen_1_out = g_model1(input_gen)
    dis_out = d_model(gen_1_out)

    # Identity Loss
    input_id = Input(shape=(SIZE, SIZE, 3))
    output_id = g_model1(input_id)

    # Cycle Loss - Forward
    output_f = g_model2(gen_1_out)

    # Cycle Loss - Backward
    gen_2_out = g_model2(input_id)
    output_b = g_model1(gen_2_out)

    # Final Model
    model = Model(
        inputs=[input_gen, input_id],
        outputs=[dis_out, output_id, output_f, output_b],
        name=name
    )
    model.compile(
        loss=['mse', 'mae', 'mae', 'mae'],
        loss_weights=[1,5,10,10],
        optimizer= Adam(learning_rate=2e-4, beta_1=0.5)
    )
    return model
```

```
def generate_real_samples(n_samples, dataset):
    ix = np.random.randint(0,dataset.shape[0], n_samples)
    X = dataset[ix]
    y = np.ones(shape=(n_samples, 8, 8, 1))
    return X, y
```

```
def generate_fake_samples(g_model, dataset):
    X = g_model.predict(dataset)
    y = np.zeros(shape=(len(dataset), 8, 8, 1))
    return X, y
```

```
def update_image_pool(pool, images, max_size=50):
    selected = list()
    for image in images:
        if len(pool) < max_size:
            pool.append(image)
            selected.append(image)
        elif random() < 0.5:
            selected.append(image)
        else:
            ix = np.random.randint(0,len(pool))
            selected.append(pool[ix])
            pool[ix] = image
    return np.asarray(selected)
```