```
!pip install -U scipy==1.2.0
from scipy.optimize import fmin_l_bfgs_b
from scipy.misc import imsave
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
import importlib
import tensorflow as tf
import tensorflow.keras as K
import math, datetime, pandas as pd, numpy as np
import matplotlib.pyplot as plt,random, pickle, glob, os
import sklearn
from PIL import Image
import tarfile
import cv2
import random
```

```
Collecting scipy==1.2.0
  Downloading scipy-1.2.0-cp37-cp37m-manylinux1_x86_64.whl (26.6 MB)
     |████████████████████████████████| 26.6 MB 75 kB/s
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: scipy
  Attempting uninstall: scipy
    Found existing installation: scipy 1.4.1
    Uninstalling scipy-1.4.1:
      Successfully uninstalled scipy-1.4.1
ERROR: pip's dependency resolver does not currently take into account all the package
jax 0.2.21 requires scipy>=1.2.1, but you have scipy 1.2.0 which is incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which
Successfully installed scipy-1.2.0
```

Loading data

```
input_size=240
```

```
path='/content/'

img_list = glob.glob(path+'**/*.jpg', recursive=True)
n = len(img_list); n
```

```
1
```

```
train_df=pd.read_csv('train_data.csv')
train_df.shape
```

```
(1, 3)
```

```
train_list=train_df.ids
```

```
len(train_list)
```

```
1
```

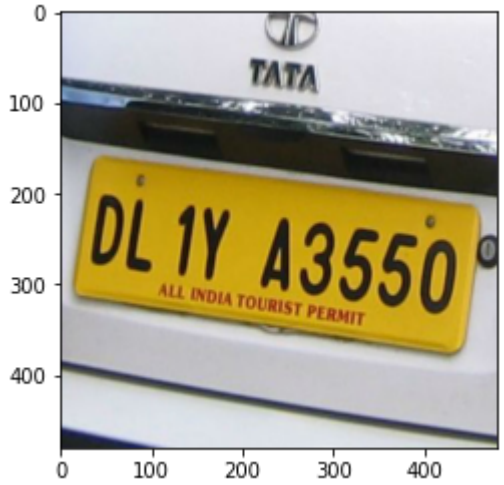
```
val_list = glob.glob('/content/*.jpg*', recursive=True)
len(val_list)
```

```
1
```

```
def read_image(img_path,scale=2):
    global input_size
    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_lr = cv2.resize(img, (input_size,input_size),interpolation=cv2.INTER_CUBIC)
    img_hr = cv2.resize(img, (input_size*scale,input_size*scale),interpolation=cv2.INTER_C
    return img_lr,img_hr
```

```
_,img=read_image(train_list[0])
print(img.shape)
plt.imshow(img)
```

```
(480, 480, 3)
<matplotlib.image.AxesImage at 0x7f53aab51a50>
```




```
_,img=read_image(val_list[0])
print(img.shape)
plt.imshow(img)
```

```
(480, 480, 3)
<matplotlib.image.AxesImage at 0x7f53a963f950>
```

Super Resolution Model

```
def conv_block(x, filters, kernel, stride=(1,1), mode='same', act=True):
    x = K.layers.Conv2D(filters, kernel_size=kernel, strides=stride, padding=mode)(x)
    return K.layers.Activation('relu')(x) if act else x
```

```
def res_block(ip,nf=16):
    x = conv_block(ip, nf, 3, (1,1))
    x = conv_block(x, nf, 3, (1,1), act=False)
    return K.layers.Add()([x,ip])


def up_block(x,nf):
    x = K.layers.UpSampling2D()(x)
    x = conv_block(x,nf,kernel=(1,1))
    return x


def get_srmodel(shape=(None,None,3)):
    inp=K.Input(shape)

    x=conv_block(inp, 16, 3, (1,1))
    for i in range(5): x=res_block(x,16)



    x=up_block(x,nf=32)
#     for i in range(2): x=res_block(x,32)

#     x=up_block(x,nf=64)
#     for i in range(1): x=res_block(x,64)

    x = K.layers.Conv2D(3,(3,3),padding='same')(x)
    return inp,x


inp,outp=get_srmodel((None,None,3))
sr_model=K.Model(inp,outp)
sr_model.summary(110)
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape            Param #      Connected
==================================================================================================
input_1 (InputLayer)            [(None, None, None, 3)] 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, None, None, 16)  448          input_1[0
__________________________________________________________________________________________________
activation (Activation)         (None, None, None, 16)  0            conv2d[0]
__________________________________________________________________________________________________
```

```
conv2d_1 (Conv2D)               (None, None, None, 16)  2320        activatic
__________________________________________________________________________
activation_1 (Activation)       (None, None, None, 16)  0           conv2d_1[
__________________________________________________________________________
conv2d_2 (Conv2D)               (None, None, None, 16)  2320        activatic
__________________________________________________________________________
add (Add)                       (None, None, None, 16)  0           conv2d_2[
                                                                    activatic
__________________________________________________________________________
conv2d_3 (Conv2D)               (None, None, None, 16)  2320        add[0][0]
__________________________________________________________________________
activation_2 (Activation)       (None, None, None, 16)  0           conv2d_3[
__________________________________________________________________________
conv2d_4 (Conv2D)               (None, None, None, 16)  2320        activatic
__________________________________________________________________________
add_1 (Add)                     (None, None, None, 16)  0           conv2d_4[
                                                                    add[0][0]
__________________________________________________________________________
conv2d_5 (Conv2D)               (None, None, None, 16)  2320        add_1[0][
__________________________________________________________________________
activation_3 (Activation)       (None, None, None, 16)  0           conv2d_5[
__________________________________________________________________________
conv2d_6 (Conv2D)               (None, None, None, 16)  2320        activatic
__________________________________________________________________________
add_2 (Add)                     (None, None, None, 16)  0           conv2d_6[
                                                                    add_1[0][
__________________________________________________________________________
conv2d_7 (Conv2D)               (None, None, None, 16)  2320        add_2[0][
__________________________________________________________________________
activation_4 (Activation)       (None, None, None, 16)  0           conv2d_7[
__________________________________________________________________________
conv2d_8 (Conv2D)               (None, None, None, 16)  2320        activatic
__________________________________________________________________________
add_3 (Add)                     (None, None, None, 16)  0           conv2d_8[
                                                                    add_2[0][
__________________________________________________________________________
conv2d_9 (Conv2D)               (None, None, None, 16)  2320        add_3[0][
__________________________________________________________________________
activation_5 (Activation)       (None, None, None, 16)  0           conv2d_9[
__________________________________________________________________________
conv2d_10 (Conv2D)              (None, None, None, 16)  2320        activatic
__________________________________________________________________________
add_4 (Add)                     (None, None, None, 16)  0           conv2d_10
                                                                    add_3[0][
__________________________________________________________________________
up_sampling2d (UpSampling2D)    (None, None, None, 16)  0           add_4[0][
__________________________________________________________________________
conv2d_11 (Conv2D)              (None, None, None, 32)  544         up_sampli
```

loading vgg for calculating perceptual loss from one of its layer

```
vgg_inp=K.Input(shape=outp.shape[1:])
vgg= K.applications.VGG16(include_top=False,
                                  input_tensor=vgg_inp)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
58892288/58889256 [==============================] - 0s 0us/step
```

```
58900480/58889256 [==============================] - 0s 0us/step
```

```
for l in vgg.layers: l.trainable=False


# preproc_layer = K.layers.Lambda(preproc)


#Here we are using vgg layer at index 36 to be the layer to calculate loss between Traget
vgg_out_layer = vgg.get_layer(index=5).output

# making model Model(inputs, outputs)
vgg_content = K.Model(vgg_inp, vgg_out_layer)

vgg_content.summary(110)
```

```
Model: "model_1"
_____________________________________________________________________________________________________
Layer (type)                                        Output Shape
=====================================================================================================
input_2 (InputLayer)                                [(None, None, None, 3)]
_____________________________________________________________________________________________________
block1_conv1 (Conv2D)                               (None, None, None, 64)
_____________________________________________________________________________________________________
block1_conv2 (Conv2D)                               (None, None, None, 64)
_____________________________________________________________________________________________________
block1_pool (MaxPooling2D)                          (None, None, None, 64)
_____________________________________________________________________________________________________
block2_conv1 (Conv2D)                               (None, None, None, 128)
_____________________________________________________________________________________________________
block2_conv2 (Conv2D)                               (None, None, None, 128)
=====================================================================================================
Total params: 260,160
Trainable params: 0
Non-trainable params: 260,160
_____________________________________________________________________________________________________
```

Data Generator and Metrics

```
def randomHorizontalFlip(img, u=0.5):
    if np.random.random() < u:
        img = cv2.flip(img, 1)
    return img
def randomVerticalFlip(img, u=0.5):
    if np.random.random() < u:
        img = cv2.flip(img, 0)
    return img


# def randomCrop(img):
#     global input_size
#     h=input_size*2
```

```
#     assert img.shape[0] >= h
#     assert img.shape[1] >= h
#     x = random.randint(0, img.shape[1] - h)
#     y = random.randint(0, img.shape[0] - h)
#     img = img[y:y+h, x:x+h]
#     return img


def train_generator():
    global batch_size
    global input_size
    while True:
        for start in range(0, len(train_list), batch_size):
            x_batch = []
            y_batch = []
            end = min(start + batch_size, len(train_list))
            ids_train_batch = train_list[start:end]
            for i,ids in enumerate(ids_train_batch):
                img = cv2.imread(ids)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                tar = cv2.resize(img, (input_size*2, input_size*2),interpolation=cv2.INTER
                tar = randomHorizontalFlip(tar)
                tar = randomVerticalFlip(tar)
                img1 = cv2.resize(tar, (input_size, input_size),interpolation=cv2.INTER_CU
                x_batch.append(img1)
                y_batch.append(tar)
            x_batch = np.array(x_batch, np.float32) / 255.
            y_batch = np.array(y_batch, np.float32) / 255.
            yield x_batch, y_batch

# i am using Set5 dataset for validation please download the data
def valid_generator():

    global batch_size
    batch_size=32
    global input_size
    while True:
        for start in range(0, len(val_list), batch_size):
            x_batch = []
            y_batch = []
            end = min(start + batch_size, len(val_list))
            ids_valid_batch = val_list[start:end]
            for i,ids in enumerate(ids_valid_batch):
                img = cv2.imread(ids)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img1 = cv2.resize(img, (input_size, input_size),interpolation=cv2.INTER_CU
                img2 = cv2.resize(img, (input_size*2, input_size*2),interpolation=cv2.INTE
                x_batch.append(img1)
                y_batch.append(img2)
            x_batch = np.array(x_batch, np.float32) / 255.
            y_batch = np.array(y_batch, np.float32) / 255.
            yield x_batch, y_batch


l=next(valid_generator())
```

```
def psnr(y_true,y_pred):
    return tf.image.psnr(y_true,y_pred,1.0)
def ssim(y_true,y_pred):
    return tf.image.ssim(y_true,y_pred,1.0)


# This is our perceptual loss function
def perceptual_loss(y_true,y_pred):
#     mse=K.losses.mean_squared_error(y_true,y_pred)
    y_t=vgg_content(y_true)
    y_p=vgg_content(y_pred)
    loss=K.losses.mean_squared_error(y_t,y_p)
    return loss


learning_rate=0.001
adam=K.optimizers.Adam(lr=learning_rate)
sr_model.compile(optimizer=adam,loss=perceptual_loss,metrics=[psnr,ssim])


batch_size=16
input_size=32


def fit(model,epoch=2):
    model.fit_generator(generator=train_generator(),
                        steps_per_epoch=np.ceil(float(len(train_list)) / float(batch_size)),
                        epochs=epoch,
                        verbose=1,
                        validation_data=valid_generator(),
                        shuffle=True,
                        validation_steps=np.ceil(float(len(val_list)) / float(batch_size)))
    return model


sr_model=fit(sr_model,10)
```

```
Epoch 1/10
1/1 [==============================] - 3s 3s/step - loss: 40.1719 - psnr: 4.8845 - ss
Epoch 2/10
1/1 [==============================] - 0s 167ms/step - loss: 36.9319 - psnr: 5.6557 -
Epoch 3/10
1/1 [==============================] - 0s 177ms/step - loss: 33.4245 - psnr: 6.0533 -
Epoch 4/10
1/1 [==============================] - 0s 194ms/step - loss: 31.6264 - psnr: 6.0062 -
Epoch 5/10
1/1 [==============================] - 0s 170ms/step - loss: 26.9437 - psnr: 6.0627 -
Epoch 6/10
1/1 [==============================] - 0s 174ms/step - loss: 24.0374 - psnr: 6.2598 -
Epoch 7/10
1/1 [==============================] - 0s 176ms/step - loss: 24.9620 - psnr: 6.4472 -
Epoch 8/10
1/1 [==============================] - 0s 187ms/step - loss: 22.2344 - psnr: 7.0352 -
Epoch 9/10
1/1 [==============================] - 0s 171ms/step - loss: 22.4084 - psnr: 7.2959 -
Epoch 10/10
1/1 [==============================] - 0s 181ms/step - loss: 21.2397 - psnr: 7.5057 -
```

progressive resizing

```
# input_size=64
# sr_model=fit(sr_model,15)


# input_size=96
# sr_model=fit(sr_model,10)


# input_size=128
# sr_model=fit(sr_model,15)


input_size=160
sr_model=fit(sr_model,10)
```

```
Epoch 1/10
1/1 [==============================] - 3s 3s/step - loss: 7.8641 - psnr: 8.2727 - ssi
Epoch 2/10
1/1 [==============================] - 3s 3s/step - loss: 7.4017 - psnr: 8.4020 - ssi
Epoch 3/10
1/1 [==============================] - 3s 3s/step - loss: 7.2271 - psnr: 8.6255 - ssi
Epoch 4/10
1/1 [==============================] - 3s 3s/step - loss: 6.6564 - psnr: 8.9619 - ssi
Epoch 5/10
1/1 [==============================] - 3s 3s/step - loss: 6.3465 - psnr: 9.6815 - ssi
Epoch 6/10
1/1 [==============================] - 3s 3s/step - loss: 5.9944 - psnr: 10.4393 - ss
Epoch 7/10
1/1 [==============================] - 3s 3s/step - loss: 5.4895 - psnr: 11.3742 - ss
Epoch 8/10
1/1 [==============================] - 3s 3s/step - loss: 5.2313 - psnr: 12.2730 - ss
Epoch 9/10
1/1 [==============================] - 3s 3s/step - loss: 5.0863 - psnr: 12.9626 - ss
Epoch 10/10
1/1 [==============================] - 3s 3s/step - loss: 4.8507 - psnr: 13.3464 - ss
```

```
input_size=224
sr_model=fit(sr_model,10)
```

```
Epoch 1/10
1/1 [==============================] - 7s 7s/step - loss: 3.1857 - psnr: 13.8676 - ss
Epoch 2/10
1/1 [==============================] - 6s 6s/step - loss: 3.0207 - psnr: 13.8468 - ss
Epoch 3/10
1/1 [==============================] - 6s 6s/step - loss: 3.0525 - psnr: 13.5905 - ss
Epoch 4/10
1/1 [==============================] - 6s 6s/step - loss: 2.9152 - psnr: 13.6006 - ss
Epoch 5/10
1/1 [==============================] - 6s 6s/step - loss: 2.6122 - psnr: 13.6593 - ss
Epoch 6/10
1/1 [==============================] - 6s 6s/step - loss: 2.6943 - psnr: 14.0638 - ss
Epoch 7/10
1/1 [==============================] - 6s 6s/step - loss: 2.5818 - psnr: 14.4705 - ss
```

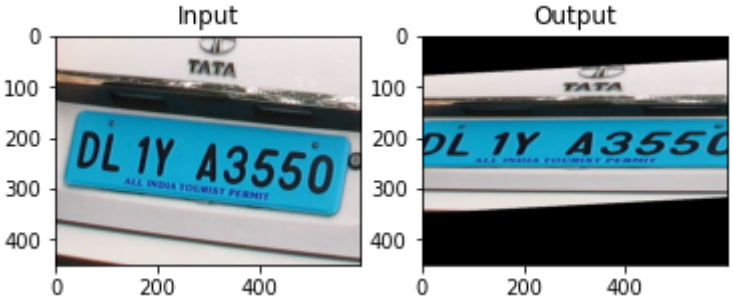
```
Epoch 8/10
1/1 [==============================] - 6s 6s/step - loss: 2.3301 - psnr: 14.8441 - ss
Epoch 9/10
1/1 [==============================] - 7s 7s/step - loss: 2.3803 - psnr: 15.3961 - ss
Epoch 10/10
1/1 [==============================] - 6s 6s/step - loss: 2.4722 - psnr: 15.7467 - ss
```

```
input_size=160


tlr,hlr=read_image(val_list[0])
tlr=np.expand_dims(tlr,axis=0)
pred=sr_model.predict(tlr/255.)


import cv2
import numpy
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
plt.axis('off')
plt.imshow(pred[0])
plt.savefig('main.jpg')
fig,axs=plt.subplots(1,3,figsize=(20,20))
ax=axs.flat

ax[0].imshow(tlr[0]/255.)
ax[1].imshow(pred[0])
ax[2].imshow(hlr/255.)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
<matplotlib.image.AxesImage at 0x7f53a2931c90>
```

```
K.models.save_model(sr_model,'sr_2x.h5')
```

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/dem.jpg')
rows, cols, ch = img.shape

pts1 = np.float32([[50, 50],
                   [200, 50],
                   [50, 200]])

pts2 = np.float32([[10, 110],
                   [200, 100],
                   [0, 200]])

M = cv2.getAffineTransform(pts1, pts2)
dst = cv2.warpAffine(img, M, (cols, rows))
cv2.imwrite('tilted.jpg',dst)
plt.subplot(121)
plt.imshow(img)
plt.title('Input')

plt.subplot(122)
plt.imshow(dst)
plt.title('Output')
plt.show()
```


Input
Output
0
100
200
300
400
0
200
400
TATA
DL 1Y A3550