Data Preparation

In [ ]:

```
# installing required Packages
!ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
from IPython.display import clear_output
clear_output(wait=True)
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices()[0]) # Cpu info
print(device_lib.list_local_devices()[3]) # Gpu info
```

```
name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 15758961345302667866

name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 15695549568
locality {
  bus_id: 1
  links {
  }
}
incarnation: 8700166528683381238
physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability
: 6.0"
```

Importing all required modules

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from numpy import asarray,zeros,moveaxis
import matplotlib.pyplot as plt
from sys import getsizeof
from tqdm import tqdm
import numpy as np
import pandas as pd
from os import path
from tensorflow.keras.initializers import *
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.optimizers import *
import os,sys,ntpath,fnmatch,shutil,cv2,gc
import joblib,time,os.path,itertools
from scipy.sparse import csc_matrix
from sklearn.utils import shuffle
from time import time
np.random.seed(0)
import warnings
warnings.filterwarnings("ignore")
from google.colab import drive
drive.mount('/content/drive')
if __name__ != '__main__':clear_output()
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4
n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_
type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis
.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

Function for Memory Utilization status

Function for Memory Utilization status

In [ ]:

```
def get_gpu_memory_status(print_status=False):

    """
    Function to print the amount of CPU and GPU Memory used at an instant
    Input  : print_status <Boolean>
    Return : Cpu memory Usage <Float>   """

    # ref: https://stackoverflow.com/questions/48750199/google-colaboratory-misleading-information-about-its-gpu-only-5-ram-available

    import psutil, os, GPUtil as GPU
    GPUs = GPU.getGPUs()  # get list of all avaliable gpus
    gpu = GPUs[0]  # first Gpu
    process = psutil.Process(os.getpid())  # Process id of current process
    s = lambda x: np.round(x / (1024**3), 2)  # lambda function to get memory in GB

    if print_status:  # print memory utilization
        print(
            "\nGen RAM Free: {0} GB  - Used: {1} GB - Total : {2} GB - Util  {3} % ".
            format(
                s(psutil.virtual_memory().available),
                s(process.memory_info().rss), s(psutil.virtual_memory().total),
                np.round(
                    s(process.memory_info().rss) * 100 / s(
                        psutil.virtual_memory().total), 2)))
        print(
            "GPU RAM Free: {0} GB  - Used: {1} GB - Total : {2} GB  - Util  {3} %  ".
            format(
                np.round(gpu.memoryFree / 1024, 2),
                np.round(gpu.memoryUsed / 1024, 2),
                np.round(gpu.memoryTotal / 1024, 2),
                np.round(gpu.memoryUtil * 100, 2)))

    return s(process.memory_info().rss)

if __name__ == '__main__': get_gpu_memory_status(True)
```

```
Gen RAM Free: 23.95 GB  - Used: 1.06 GB - Total : 25.51 GB - Util  4.16 %
GPU RAM Free: 15.55 GB  - Used: 0.34 GB - Total : 15.9 GB  - Util  2.17 %
```

Data Generator Choice

In [ ]:

```
if __name__ == '__main__': # get user choice
    root='/content/drive/My Drive/Colab Notebooks/'
    Choice=int(input("Enter your choice for Data Generator (From where should be the Data loaded for each Batch During Training): \n\n\
    1.Raw Image Files from Disk(High Time complexity)  2.Default(Recommended)  3.Saved Joblib Data(High Space complexity):\n\n>>>"))
    Dump=joblib.dump(Choice,root+"Choice")
```

```
Enter your choice for Data Generator (From where should be the Data loaded for each Batch During Training)
:

1.Raw Image Files from Disk(High Time complexity)  2.Default(Recommended)  3.Saved Joblib Data(High Space
complexity):

>>>2
```

Get list of all data file names of Part1

In [ ]:

```
# path of data files Part1
root='/content/drive/My Drive/Colab Notebooks/'
train_img_path1=root+'IDD_Segmentation/leftImg8bit/train/'
train_label_path1=root+'IDD_Segmentation/gtFine/train_label_level1/'
train_img_path2=root+'idd20kII/leftImg8bit/train/'
train_label_path2=root+'idd20kII/gtFine/train_label_level1/'

# list files of data files Part1
train_img_files1=sorted(os.listdir(root+'IDD_Segmentation/leftImg8bit/train'))
train_label_files1=sorted(os.listdir(root+'IDD_Segmentation/gtFine/train_label_level1'))
val_img_files1=sorted(os.listdir(root+'IDD_Segmentation/leftImg8bit/val'))
val_label_files1=sorted(os.listdir(root+'IDD_Segmentation/gtFine/val_label_level1'))
```

Get list of all data file names of Part2

In [ ]:

```
# path of data files Part1
root='/content/drive/My Drive/Colab Notebooks/'
val_img_path1=root+'IDD_Segmentation/leftImg8bit/val/'
val_label_path1=root+'IDD_Segmentation/gtFine/val_label_level1/'
val_img_path2=root+'idd20kII/leftImg8bit/val/'
val_label_path2=root+'idd20kII/gtFine/val_label_level1/'

# list files of data files Part1
train_img_files2=sorted(os.listdir(root+'idd20kII/leftImg8bit/train'))
train_label_files2=sorted(os.listdir(root+'idd20kII/gtFine/train_label_level1'))
val_img_files2=sorted(os.listdir(root+'idd20kII/leftImg8bit/val'))
val_label_files2=sorted(os.listdir(root+'idd20kII/gtFine/val_label_level1'))
```

Image Data preparation

In [ ]:

```
def Prepare_Image_and_Save(path, name, img_files):
    '''
    ------------------------------------------------------------
    Function to prepare Image data for a given data files
        1. Read all images one after another from specifed Directory.
        2. Resize images after reading it to some height and width
        3. Normalize the pixel values in image by dividing by 255
        4. Save Prepared data for future Usage
    ------------------------------------------------------------
        Parameters
        ----------
        path <list of Path>     : List of Absolute Path of images Data files.
        name <String>           : File name to save prepared data
        img_files <List>        : List of Image data file names

        returns
        --------
        Boolean<True>  : Indicate successful Data Preparation
    ------------------------------------------------------------

    '''
    height, width, n_classes = 240, 480, 7
    image = []
    for j in range(len(path)):
        for i in tqdm(range(len(img_files[j]))):
            img = cv2.imread(path[j] + img_files[j][i])
            img = cv2.resize(img, (width, height))
            img = np.float32(img) / 255
            image.append(img)
    print(len(image))
    joblib.dump(image, name)
    return True

# prepare image based on Data generator choice
path_s = "/content/drive/My Drive/prep_train_img_files_save_80"
if __name__ == '__main__' and Choice>1 :
    path1 = [root+'IDD_Segmentation/leftImg8bit/train/',root+'idd20kII/leftImg8bit/train/']
    path2 = [root+'IDD_Segmentation/leftImg8bit/val/',root+'idd20kII/leftImg8bit/val/']
    img_files1,img_files2 = [train_img_files1,train_img_files2],[val_img_files1,val_img_files2]

    Indicator1 = Prepare_Image_and_Save(path1,"/content/drive/My Drive/Colab Notebooks/prep_train_img_files_save",img_files1)
    Indicator2 = Prepare_Image_and_Save(path2,"/content/drive/My Drive/Colab Notebooks/prep_val_img_files_save",img_files2)
    if Indicator1 and Indicator2: print("Data Preparation of Images Successful Done!")

elif __name__ != '__main__':
    #if not path.exists(path_s):raise Exception("!File not Found: First Run IID_Data_Prep_Utils.ipynb")
    print("Checking Status:\n"+"-"*54+"\n1.Image Data Preparation     .. .. .. >>> |Done| <1/5>")
```

```
Data Preparation of Images Successful Done!
```

Label Mask Data preparation

In [ ]:

```
def Prepare_Label_and_Save(path, n_classes, name):
```

```
def Prepare_Label_and_Save(path, n_classes, name):
    '''
    -------------------------------------------------------------
    Function to prepare Label data for a given data files
        1. Read all Mask one after another from specifed Directory.
        2. Resize Mask after reading it to some height and width
        3. Performing one hot ecoding on mask resulting in 3D matrix
        4. Save Prepared data in sparse representation for future Usage
    -------------------------------------------------------------
        Parameters
        ----------
        path <list of Path>     : List of Absolute Path of images Data files.
        n_classes <Integer>     : Number of classes in Mask
        name <String>           : File name to save prepared data

        returns
        --------
        Boolean<True>  : Indicate successful Data Preparation
    -------------------------------------------------------------

    '''
    sparse_list = []
    for k in range(len(path)):
        files = sorted(os.listdir(path[k]))
        height, width, n_classes = 240, 480, n_classes
        for j in tqdm(range(len(files))):
            label = np.zeros((n_classes, height, width), dtype=np.uint8)
            img = cv2.imread(path[k] + "/" + files[j], cv2.IMREAD_GRAYSCALE)
            img1 = cv2.resize(img, (width, height))
            for i in range(n_classes):
                label[i, :, :] = (img1 == i).astype(np.uint8)
            sp_list = []
            for i in range(label.shape[0]):
                sp_list.append(csc_matrix(label[i]))
            sparse_list.append(sp_list)
    joblib.dump(sparse_list, name)
    return True

# prepare Mask based on Data generator choice
if __name__ == '__main__' and Choice>1 :

    path1=[root+"IDD_Segmentation/gtFine/train_label_level1",root+"idd20kII/gtFine/train_label_level1"]
    path2=[root+"IDD_Segmentation/gtFine/val_label_level1",root+"idd20kII/gtFine/val_label_level1"]
    Indicator=Prepare_Label_and_Save(path1,7,root+"data/prep_train_label_files_save1")
    Indicator=Prepare_Label_and_Save(path2,7,root+"data/prep_val_label_files_save1")
    if Indicator1 and Indicator2: print("Data Preparation of Labels Successful Done!")

elif __name__ != '__main__':print("2.Label Mask Preparation     .. .. .. >>> |Done| <2/5>")
```

```
Data Preparation of Labels Successful Done!
```

Shuffle prepared Data Samples

In [ ]:

```
def Shuffle_Data(root="/content/drive/My Drive/"):
    '''
    Function to shuffle data to get rid bias
    Input
    root<String>  : Absolute Path of Data where it is saved.

    Return
    <Boolean>     : True Indicate successful Data Shuffle
    '''

    # Shuffle Prepared Image Data
    path1, path3=root+"data/prep_train_img_files_save", root+"data/prep_train_label_files_save1"
    if path.exists(path1) and path.exists(path3):
        prep_train_img_files_save, prep_train_label_files_save=shuffle(joblib.load(path1),joblib.load(pat
h3),random_state=0)
        Dump=joblib.dump(prep_train_img_files_save, path1), joblib.dump(prep_train_label_files_save, path
3)
        del prep_train_img_files_save, prep_train_label_files_save
        Junk= gc.collect()

    # Shuffle Prepared Label Mask Data
    path2, path4=root+"data/prep_val_img_files_save", root+"data/prep_val_label_files_save1"
    if path.exists(path2) and path.exists(path4):
        prep_val_img_files_save, prep_val_label_files_save=shuffle(joblib.load(path2),joblib.load(path4),
random_state=0)
        Dump=joblib.dump(prep_val_img_files_save, path2), joblib.dump(prep_val_label_files_save, path4)
        del prep_val_img_files_save, prep_val_label_files_save
```

```
        Junk= gc.collect()

    return True

# Invoke based on choice
if __name__ == '__main__' and Choice>1 :
    Indicator=Shuffle_Data()
    if Indicator:print("Data Shuffle Successful Done!")

elif __name__ != '__main__':print("3.Data Shuffling             .. .. .. >>> |Done| <3/5>")
```

```
Data Shuffle Successful Done!
```

Train Test split on data

In [ ]:

```
def Train_Test_Split(root="/content/drive/My Drive/", split=0.8):
    '''
    Function to perfrom Train test split of data

    Input
    root<String>   : Absolute Path of Data where it is saved.
    split<Float>   : Value specify split size of train data

    Return
    <Boolean>      : True Indicate successful Train Test Split
    '''

    total_no_samples=14027
    split_index=(int((total_no_samples*0.8)/32))*32

    path1, path3= root+"data/prep_train_img_files_save", root+"data/prep_train_label_files_save1"
    prep_train_img_files_save, prep_train_label_files_save=joblib.load(path1), joblib.load(path3)

    joblib.dump(prep_train_img_files_save[:split_index], root+"/prep_train_img_files_save_80")
    joblib.dump(prep_train_label_files_save[:split_index], root+"/prep_train_label_files_save_80")
    joblib.dump(prep_train_img_files_save[split_index:], root+"/prep_test_img_files_save_20")
    joblib.dump(prep_train_label_files_save[split_index:], root+"/prep_test_label_files_save_20")
    clear_output()
    return True

# Invoke based on choice
if __name__ == '__main__' and Choice>1 :
    Indicator=Train_Test_Split()
    if Indicator:print("Data Train_Test_Split Successful Done!")

elif __name__ != '__main__':print("4.Data Train_Test_Split      .. .. .. >>> |Done| <4/5>")
```

```
Data Train_Test_Split Successful Done!
```

Data Generator to generate samples from raw data

In [ ]:

```
# Generate data from disk which is prepared along with each batch during Training
height,width,n_classes=240,480,7
def image_generator_d(data,batch_files,start,end):
    '''
    Function to prepare data and generate samples for each batch during training
    This function is called recursively for each batch in each epoch

    Input
    data<String>         : Absolute Path of Data Directory
    batch_files<Float>   : Contain data files names for a batch
    start<Integer>       : Start index value of sample for that batch
    end <Integer>        : End index value of sample for that batch

    Return
    image<Array>      : Data matrix for that Batch
    '''
    image=[]
    for i in range(len(batch_files)):
        img = cv2.imread(data+batch_files[i])
        img = cv2.resize(img,(width,height))
        img = np.float32(img)/255
        image.append(img)
    image=np.array(image)
    return image
```

```
def label_generator_d(data,batch_files,start,end):
    '''
    Function to prepare Label mask and generate samples for each batch during training
    This function is called recursively for each batch in each epoch

    Input
    data<String>          : Absolute Path of Mask Directory
    batch_files<Float>    : Contain data files names for a batch
    start<Integer>        : Start index value of sample for that batch
    end <Integer>         : End index value of sample for that batch

    Return
    labels<Array>       : Label matrix for that Batch
    '''
    labels=[]
    for i in range(len(batch_files)):
        label = np.zeros((height, width, n_classes)).astype(np.uint8)
        img = cv2.imread(data+batch_files[i])
        img = cv2.resize(img,(width,height))
        img1 = img[:,:,0]
        for i in range(n_classes):
            label[:,:,i] = (img1==i).astype(np.uint8)
        labels.append(label)
    labels=np.array(labels)
    return labels

def train_batch_generator_d(batch_size,epochs):
    '''
    Function to get prepared train data for training for each batch
    This function is recursively calls image and label generator for each batch

    Input
    epochs<Integer>       : Number of epochs to train
    batch_size<Integer>   : Training batch size

    Return
    (batch_x, batch_y)    : Yield images and labels for each batch
    '''

    global train_img_path1,train_label_path1, train_img_path2,train_label_path2
    tr_L = len(train_img_files1)+len(train_img_files2)  # number of total data files
    num_tr=0

    while num_tr<epochs*2 :    # while loop to run for specified epoch
        train_batch_start=0
        train_batch_end = batch_size

        while train_batch_start < tr_L:    # while loop to run for Total number of samples
            train_limit = min(train_batch_end, tr_L)

            if train_limit<len(train_img_files1):    # Get train data from part1
                train_img_path,train_img_files,train_label_path,train_label_files,train_offset_start,trai
n_offset_limit=train_img_path1,train_img_files1,train_label_path1,train_label_files1,0,0

            elif train_limit>len(train_img_files1) and train_limit<(len(train_img_files1)+batch_size):
# Get train data from end of part1 and start of part2
                train_limit,train_batch_start,train_offset_start,train_offset_limit=len(train_img_files1
),len(train_img_files1)-batch_size,0,0

            elif train_limit>len(train_img_files1) and train_limit>=(len(train_img_files1)+batch_size):
# Get train data from part2
                train_img_path,train_img_files,train_label_path,train_label_files=train_img_path2,train_i
mg_files2,train_label_path2,train_label_files2
                train_offset_start=train_offset_limit=len(train_img_files)

            # Call Image and label Generator for batch_size number of prepared train Samples
            batch_x = image_generator_d(train_img_path,train_img_files[train_batch_start-train_offset_sta
rt:(train_batch_start-train_offset_start+batch_size)],train_batch_start-train_offset_start,(train_batch_st
art-train_offset_start+batch_size))
            batch_y = label_generator_d(train_label_path,train_label_files[train_batch_start-train_offset
_start:(train_batch_start-train_offset_start+batch_size)],train_batch_start-train_offset_start,(train_bat
ch_start-train_offset_start+batch_size))
            yield (batch_x,batch_y)          # yield X,Y for training for each batch
            train_batch_start += batch_size
            train_batch_end += batch_size    # reinitialize start and end for next batch
        num_tr+=1

def val_batch_generator_d(batch_size,epochs):
    '''
    Function to get prepared Validation data for training for each batch
    This function is recursively calls image and label generator for each batch in each epoch
```

```
    Input
    epochs<Integer>       : Number of epochs to train
    batch_size<Integer>   : Training batch size

    Return
    (batch_x, batch_y)    : Yield images and labels for each batch
    '''
    global val_img_path1,val_label_path1, val_img_path2,val_label_path2
    val_L = len(val_img_files1)+len(val_img_files2)  # number of total val data files
    num_val=0

    while num_val<epochs*2:    # while loop to run for specified epoch
        val_batch_start,val_batch_end=0,batch_size

        if get_gpu_memory_status()>25.25:
            get_gpu_memory_status(True)    # raise error if RAM is almost full
            raise Exception("Ram Almost Full")

        while val_batch_start < val_L:    # while loop to run for Total number of samples
            val_limit = min(val_batch_end, val_L)

            if val_limit<len(val_img_files1):    # Get val data from part1
                val_img_path,val_img_files,val_label_path,val_label_files,val_offset_start,val_offset_lim
it=val_img_path1,val_img_files1,val_label_path1,val_label_files1,0,0

            elif val_limit>len(val_img_files1) and val_limit<(len(val_img_files1)+batch_size):  # Get val
data from end of part1 and start of part2
                val_limit,val_batch_start,val_offset_start,val_offset_limit=len(val_img_files1),len(val_i
mg_files1)-batch_size,0,0

            elif val_limit>len(val_img_files1) and val_limit>=(len(val_img_files1)+batch_size): # Get val
data from part2
                val_img_path,val_img_files,val_label_path,val_label_files=val_img_path2,val_img_files2,va
l_label_path2,val_label_files2
                val_offset_start=val_offset_limit=len(val_img_files1)

            # Call Image and label Generator for batch_size number of prepared val Samples
            batch_valx = image_generator_d(val_img_path,val_img_files[val_batch_start-val_offset_start:(v
al_batch_start-val_offset_start+batch_size)],val_batch_start-val_offset_start,(val_batch_start-val_offset
_start+batch_size))
            batch_valy = label_generator_d(val_label_path,val_label_files[val_batch_start-val_offset_star
t:(val_batch_start-val_offset_start+batch_size)],val_batch_start-val_offset_start,(val_batch_start-val_of
fset_start+batch_size))
            yield (batch_valx, batch_valy)  # yield X,Y for validation each batch
            val_batch_start += batch_size
            val_batch_end += batch_size     # reinitialize start and end for next batch
        num_val+=1
```

Data Generator to generate samples from prepared saved joblib file

In [ ]:

```
# Generate data for each batch from where data already prepared and saved as joblib File
height,width,n_classes=240,480,7
def image_generator_j(start,end,saved_data):
    '''Function to get samples from saved data for each batch in each epoch'''
    return np.array(saved_data[start:end])

def label_generator_j(start,end,saved_data):
    '''Function to get Mask from saved data for each batch in each epoch'''
    slice_saved=saved_data[start:end]
    ar=np.empty((len(slice_saved),n_classes,height,width), dtype=np.uint8)
    for j in range(len(slice_saved)):
        for i in range(n_classes):
            ar[j][i]=slice_saved[j][i].todense()
    ar = moveaxis(ar, 1, 3)
    return ar

def train_batch_generator_j(batch_size,epochs):
    '''Function to yield train Images and Labels for each batch from saved Joblib file'''
    global prep_train_img_files_save, prep_train_label_files_save
    tr_L = len(prep_train_img_files_save)
    num_tr=0
    while num_tr<epochs*2 :
        train_batch_start=0
        train_batch_end = batch_size
        while train_batch_start < tr_L:
            train_limit = min(train_batch_end, tr_L)
            batch_x = image_generator_j(train_batch_start,train_limit,prep_train_img_files_save)
            batch_y = label_generator_j(train_batch_start,train_limit,prep_train_label_files_save)
            yield (batch_x,batch_y)
```

```
            train_batch_start += batch_size
            train_batch_end += batch_size
        num_tr+=1

def val_batch_generator_j(batch_size,epochs):
    '''Function to yield validation Images and Labels for each batch from saved Joblib file'''
    global prep_val_img_files_save, prep_val_label_files_save
    val_L = len(prep_val_img_files_save)
    num_val=0
    while num_val<epochs*2:
        val_batch_start=0
        val_batch_end = batch_size
        while val_batch_start < val_L:
            val_limit = min(val_batch_end, val_L)
            batch_valx = image_generator_j(val_batch_start,val_limit,prep_val_img_files_save)
            batch_valy = label_generator_j(val_batch_start,val_limit,prep_val_label_files_save)
            yield (batch_valx,batch_valy)
            val_batch_start += batch_size
            val_batch_end += batch_size
        num_val+=1
```

Function to Compute Mean Intersection-Over-Union (MIOU) during training

In [ ]:

```
def Calculate_MIOU(y_val, y_pred):
    '''
    Function to compute Mean Intersection-Over-Union (MIOU)
    MIOU is the average of all Intersection-Over-Union (IOU = true_positive / (true_positive + false_posit
ive + false_negative)) over all classes

    Input
    y_val<ndarray>      : True samples
    y_pred<ndarray>     : Predicted Samples

    Return
    MIoU<Float>         : MIOU Score

    '''
    class_iou ,n_classes=[],7
    y_predi = np.argmax(y_pred, axis=3)
    y_truei = np.argmax(y_val, axis=3)
    for c in range(n_classes):
        TP = np.sum((y_truei == c) & (y_predi == c))
        FP = np.sum((y_truei != c) & (y_predi == c))
        FN = np.sum((y_truei == c) & (y_predi != c))
        IoU = TP / float(TP + FP + FN)
        if(float(TP + FP + FN) == 0):
            IoU=TP/0.001
        class_iou.append(IoU)
    MIoU=sum(class_iou)/n_classes
    return MIoU

def miou( y_true, y_pred ) :
    '''Funtion to Wraps a miou function into a TensorFlow op that executes when needed'''
    score = tf.py_function( lambda y_true, y_pred : Calculate_MIOU( y_true, y_pred).astype('float32'),[y
_true, y_pred],'float32')
    return score
```

Function to Configure Data based on Choice

In [ ]:

```
# Configure function and data based on Data Generator Choice
root="/content/drive/My Drive/"
prep_train_img_files_save=prep_val_img_files_save=prep_train_label_files_save=prep_val_label_files_save=No
ne
Choice=joblib.load(root+"/Colab Notebooks/Choice")

if Choice==1 and __name__ != '__main__':
    train_batch_generator, val_batch_generator = train_batch_generator_d, val_batch_generator_d

elif Choice==2 and __name__ != '__main__':
    train_batch_generator, val_batch_generator = train_batch_generator_j, val_batch_generator_d
    prep_train_img_files_save, prep_train_label_files_save = joblib.load(root+"/prep_train_img_files_save
_part1"),joblib.load(root+"/prep_train_label_files_save_part1")

elif Choice==3 and __name__ != '__main__':
    train_batch_generator, val_batch_generator = train_batch_generator_j, val_batch_generator_j
```

```
    prep_train_img_files_save, prep_train_label_files_save = joblib.load(root+"/prep_train_img_files_save
_part1"), joblib.load(root+"/prep_train_label_files_save_part1")
    prep_val_img_files_save, prep_val_label_files_save = joblib.load(root+"data/prep_val_img_files_save")
, joblib.load(root+"data/prep_val_label_files_save1")

if __name__ != '__main__':print("5.Loading Final Data          .. .. .. >>> |Done| <5/5>\n"+"-"*54)
```

Invoke Garbage Collector

In [ ]:

```
# Get memory status and Invoke Garbage Collector
res = get_gpu_memory_status(True)
collected = gc.collect()
```