## Importing all required modules

In [ ]:

```
# all imports
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from numpy import asarray,zeros,moveaxis
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
import joblib,time,os.path,itertools
import matplotlib.pyplot as plt
from scipy.sparse import csc_matrix
from sklearn.metrics import *
from IPython.display import clear_output
from tqdm import tqdm_notebook,tqdm
from keras.models import load_model
np.random.seed(0)
import warnings
warnings.filterwarnings("ignore")
clear_output()
```

## Utility Functions

In [ ]:

```
def Select_Model(weights_save_path=False):

    """
    Function to select a Model based on entered choice and load its respective Model weights for Successfu
l Prediction
    Input:  weights_save_path <Saved weights path or Boolean>
    Returns: Model <Model Object>       """

    import ntpath,segmentation_models as sm

    m=int(input("    1.Restnet_50+U-Net    2.Unet    3.DeeplabV3    4.Pspnet    5.Segnet >> Enter a numb
er to Choose a Model:\n>>> "))

    if m==1:

        if weights_save_path==False:weights_save_path="/content/drive/My Drive/IID_Files1/New_Model_logs_
save/Unet_imgnet_resnet50_nlrr.hdf5"
        sent, sm_weight=ntpath.basename(weights_save_path)[12:],""
        for ch in sent:
            if((ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z')) or (ch >= '0' and ch <= '9'):
                sm_weight+=ch
            else: break

        Model = sm.Unet(sm_weight,classes=7,input_shape=(224, 480,3),activation='softmax')
        Model.load_weights(weights_save_path) if isinstance(weights_save_path, str) else Model.load_weigh
ts("/content/drive/My Drive/IID_Files1/New_Model_logs_save/Unet_imgnet_resnet50_nlrr.hdf5")

    elif m==2:
        Model = Unet_Segmentation((240, 480,3), 7)
        Model.load_weights(weights_save_path) if isinstance(weights_save_path, str) else Model.load_weigh
ts("/content/drive/My Drive/IID_Files/Models_save/Unet.best.hdf5")


    elif m==3:
        Model = Deeplab_V3((240, 480,3), 7)
        Model.load_weights(weights_save_path) if isinstance(weights_save_path, str) else Model.load_weigh
ts("/content/drive/My Drive/IID_Files/Models_save/Deeplab.best.hdf5")

    elif m==4:
        Model=PSPNET((240, 480,3), 7)
        Model.load_weights(weights_save_path) if isinstance(weights_save_path, str) else Model.load_weigh
ts("/content/drive/My Drive/IID_Files/Models_save/Pspnet.best.hdf5")
```

```
    elif m==5:
        Model=Segnet_Segmentation((240,480,3), 7, 1)
        Model.load_weights(weights_save_path) if isinstance(weights_save_path, str) else Model.load_weigh
ts("/content/drive/My Drive/IID_Files/Models_save/Segnet.best.hdf5")

    else: raise Exception("Wrong Input>>Enter an Integer to choose a Model")

    return Model

def plot_segmentation(images, pred_labels, true_labels=[], plot_limit=5):

    """
    Function to plot Sub-plot of Image, Label and Model Output after prediction is performed
    Input  : (images, pred_labels, true_labels) <3dArray>, plot_limit <Int>
    Return : None """

    if (len(true_labels)==0):
        for i in range(images.shape[0]):
            if i==plot_limit:return
            plt.figure(figsize=(14, 10))
            plt.subplot(1, 2, 1)
            plt.imshow(cv2.cvtColor(images[i],cv2.COLOR_BGR2RGB))
            plt.ylabel('Image',fontsize=16)
            plt.subplot(1, 2, 2)
            plt.imshow(color_code(pred_labels[i]))
            plt.ylabel('Prediction',fontsize=16)
            plt.tight_layout()
            plt.show()
    else:
        for i in range(images.shape[0]):
            if i==plot_limit:
                print("  "+" ------------------ "*8)
                return
            plt.figure(figsize=(28, 20))
            plt.subplot(1,3,1)
            plt.imshow(cv2.cvtColor(images[i],cv2.COLOR_BGR2RGB))
            plt.ylabel('Image',fontsize=28)
            plt.subplot(1,3,2)
            plt.imshow(color_code(true_labels[i]))
            plt.ylabel('Label',fontsize=28)
            plt.subplot(1,3,3)
            plt.imshow(color_code(pred_labels[i]))
            plt.ylabel('Prediction',fontsize=28)
            plt.tight_layout()
            plt.show()


def plot_confusion_matrix(cm, normalize=True, title='Confusion matrix', cmap=plt.cm.Reds):

    ''' Function to plot Confusion Matrix for given 2D_Matrix '''

    # ref: https://github.com/scikit-learn/scikit-learn/issues/12700

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    classes_dict={'Drivable': 0, 'Non Drivable': 1, 'Living Things': 2, 'Vehicles': 3, 'Road Side Object
s': 4, 'Far Objects': 5, 'Sky': 6}
    classes=list(list(classes_dict.keys()))
    plt.figure(figsize=(8,8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title,fontsize=16)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90,fontsize=12)
    plt.yticks(tick_marks, classes,fontsize=12)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label',fontsize=18)
    plt.xlabel('Predicted label',fontsize=18)
    plt.show()


def Print_result(Mean_MIoU, cf_matrix, Accuracy, true_labels_org, pred_labels_org, cr=True):
```

```
    ''' Function to print all Final computed Results '''

    # printing results
    print("\nPrinting Results:>>\n")
    print('-----------------------')
    print('|      MIOU Score     |')
    print('-----------------------')
    print('\n   MIOU Score: {}\n'.format(np.round(np.mean(Mean_MIoU),4)))

    print('-----------------------')
    print('|   Accuracy Score    |')
    print('-----------------------')
    print('\n   Accuracy Score: {}\n'.format(np.round(np.mean(Accuracy),4)))

    print('-----------------------')
    print('|  Confusion Matrix   |')
    print('-----------------------')
    plot_confusion_matrix(cf_matrix,normalize=True)

    if cr==True:
        print('--------------------------')
        print('| Classifiction Report   |')
        print('--------------------------')
        print("\n",classification_report(true_labels_org.ravel(),pred_labels_org.ravel()))
        print({'Drivable': 0, 'Non Drivable': 1, 'Living Things': 2, 'Vehicles': 3, 'Road Side Objects':
4, 'Far Objects': 5, 'Sky': 6})

    return

def Load_For_Prediction(String):

    """
    Function to Load Serialized preprocessed Data of Train, Val and Test based on choice for Prediction
    Input  : String <String>
    Return : prep_train_img_files_save <X>, prep_train_label_files_save <Y> """

    global prep_train_img_files_save, prep_train_label_files_save

    if String =="Train_data":
        prep_train_img_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep_t
rain_img_files_save_part1")
        prep_train_label_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep
_train_label_files_save_part1")
    elif String =="Val_data":
        prep_train_img_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep_v
al_img_files_save")
        prep_train_label_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep
_val_label_files_save1")
    elif String =="Test_data":
        prep_train_img_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep_t
rain_img_files_save_part2")
        prep_train_label_files_save=joblib.load("/content/drive/My Drive/IID_Files/Data_Save_240*480/prep
_train_label_files_save_part2")
    else:
        raise Exception("Enter one of the string\n'Train_data' or 'Val_data' or 'Test_data'")

    return prep_train_img_files_save, prep_train_label_files_save

def plot_training_result(out):

    ''' Function to plot Epoch vs Crossentropy and Epoch vs MIOU graph after Traning of a Model '''

    # Sub_plot with two figures
    figure, loc_ind = plt.subplots(1, 2,figsize=(15,5))

    # Obtain values to plot
    miou, val_miou= out.history['miou'], out.history['val_miou']
    loss, val_loss= out.history['loss'], out.history['val_loss']

    # epoch list
    num_epochs = list(range(1,len(miou)+1))

    # ploting Epoch vs Crossentropy Loss
    loc_ind[0].plot(num_epochs,loss,'r',label='train loss',linewidth=1.25)
    loc_ind[0].plot(num_epochs,val_loss,'g',label='validation loss',linewidth=1.25)
    loc_ind[0].set_ylabel('Categorical Crossentropy Loss',fontsize=14)
    loc_ind[0].set_xlabel('Epoch',fontsize=14)
    loc_ind[0].set_title('Epoch vs Crossentropy Loss',fontsize=16)
    loc_ind[0].grid()
    loc_ind[0].legend()

    # Epoch vs Mean Intersection over union
    loc_ind[1].plot(num_epochs,miou,linestyle='--',marker='o',color="deeppink",label='train miou',linewid
```

```
th=1.25)
    loc_ind[1].plot(num_epochs,val_miou,linestyle='--',marker='o',color="dodgerblue",label='val miou',lin
ewidth=1.25)
    loc_ind[1].set_ylabel('Mean Intersection over union',fontsize=14)
    loc_ind[1].set_xlabel('Epoch',fontsize=14)
    loc_ind[1].set_title('Epoch vs Mean Intersection over union',fontsize=16)
    loc_ind[1].grid()
    loc_ind[1].legend()
    plt.tight_layout(pad=3.0)
    plt.show()

def color_code(le_pred):

    """
    Function to Color Code given Label Mask using RGB-Color to make it suitable to Display
    Input  : le_pred <2D_Array>
    Return : col_pred <3d_Array>   """

    col_pred = moveaxis(np.repeat(le_pred[:, :, np.newaxis], 3, axis=2), -1, 0)
    color_code=[[128, 64, 128], [72, 98, 91], [255, 204, 54], [220, 20, 60], [147, 114, 178], [132, 91,
83], [70, 70, 70], [105, 143, 35]]
    m, n= col_pred.shape[1], col_pred.shape[2]
    for k in range(3):
        for i in range(m):
            for j in range(n):
                index=7 if int(col_pred[k][i][j])==255 else int(col_pred[k][i][j])
                col_pred[k][i][j]=color_code[index][k]
    col_pred=np.moveaxis(col_pred, 0, -1)
    return col_pred


def image_prepare(data,batch_files,Model):

    '''Read and preprocess images to generete batch of images for prediction'''

    height,width,n_classes,image=Model.input_shape[1],480,7,[]
    for img_i in range(len(batch_files)):
        img = cv2.imread(data+batch_files[img_i])
        img = cv2.resize(img,(width,height))
        img = np.float32(img)/255
        image.append(img)
    image=np.array(image)
    return image


def image_prepare_j(slice_saved,Model):
    '''Function to get samples batch wise from saved data for Prediction'''
    return np.array(slice_saved)


def label_prepare_j(slice_saved,Model):

    '''Function to get Mask batch wise from saved data for Prediction'''

    height,width,n_classes=Model.input_shape[1], 480, 7
    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 label_prepare(data,batch_files,Model):

    '''Read and preprocess Label Mask to generete batch of Labels for prediction'''

    height,width,n_classes,labels=Model.input_shape[1],480,7,[]
    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 prob_to_label(cf_matrix, Accuracy, predictions, label=False):

    ''' Function to get Actual Labels from Predicted probabilities to compute confusion matrix and Accurac
```

```
y '''

    pred_Out,true_Out,pred_labels,true_labels=[],[],[],[]
    predictions = moveaxis(predictions[np.newaxis,:, :,:] if (len(predictions.shape)==3) else predictions
, 3, 1)
    if isinstance(label,np.ndarray):label = moveaxis(label[np.newaxis,:, :,:] if (len(label.shape)==3) el
se label, 3, 1)

    for p_index in range(predictions.shape[0]):

        p1 = np.where(predictions[p_index]<0.5, predictions[p_index], 1)# if a[ijk]>=0.5 then a[ijk]=1 (F
alse)
        p1 = np.where(p1==1, p1, 0).astype(int)
        w = np.argmax(p1, axis=0).astype(int) # 248*480 matrix with 1-6

        if isinstance(label,np.ndarray):

            p2= np.where(label[p_index]==1, label[p_index], 0)
            v = np.argmax(label[p_index], axis=0).astype(int)    # 240*480
            Accuracy.append(np.round(accuracy_score(v.ravel(), w.ravel()),4))

            cf_matrix=np.add(cf_matrix, confusion_matrix(v.ravel(), w.ravel(),labels=[0,1,2,3,4,5,6]))
            true_labels.append(v),true_Out.append(p2)

        pred_labels.append(w),pred_Out.append(p1)

    if isinstance(label,np.ndarray):true_Out=moveaxis(np.array(true_Out), 1, 3)
    pred_Out=moveaxis(np.array(pred_Out), 1, 3)

    return pred_Out, pred_labels, true_Out, true_labels, cf_matrix, Accuracy


def Intersect_over_union(y_val, y_pred, Mean_MIoU):

    """
    Function to compute Intersect_Over_Union for given set of Samples Inputs
    Input  : y_val <4D_Array>, y_pred <4D_Array>, Mean_MIoU <List>
    Return : Mean_MIoU <List>       """

    for index in range(y_pred.shape[0]):

        class_iou ,n_classes=[],7
        y_predi = np.argmax(y_pred[index], axis=2)
        y_truei = np.argmax(y_val[index], axis=2)

        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 / (TP + FP + FN) if (TP + FP + FN)>0 else 0
            class_iou.append(IoU)

        MIoU=sum(class_iou)/n_classes
        Mean_MIoU.append(MIoU)

    return Mean_MIoU
```

### Unet Definition for prediction

In [ ]:

```
def Unet_Segmentation(input_shape, n_classes):

    def Unet_En_Blocks(Block_Number, Name, Filters, Kernel_Size, Pool_size, Previous_layer, initialize="h
e_normal"):

        MaxPool = tf.keras.layers.MaxPooling2D(pool_size=Pool_size, name= Name+"_Maxpool")(Previous_layer
) if Block_Number>1 else Previous_layer
        Convolution1 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv1", activation = 'r
elu', kernel_initializer= initialize, padding='same')(MaxPool)
        Convolution2 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv2", activation = 'r
elu', kernel_initializer= initialize, padding='same')(Convolution1)

        return Convolution2

    def Unet_Dec_Blocks(Block_Number, Name, Filters, Kernel_Size, Previous_layer, Layer_to_Concatenate, i
nitialize="he_normal"):

        Up_Sample = tf.keras.layers.UpSampling2D(size=(2, 2), name= Name+"_Upsample")(Previous_layer)
        Up_Convolution = tf.keras.layers.Conv2D(Filters,(2,2), name= Name+"_UpConv", activation = 'relu'
, padding = 'same',kernel_initializer=initialize)(Up_Sample)
```

```
        Concatenated_Layer=tf.keras.layers.Concatenate(axis=3, name= Name+"_Concat")([Layer_to_Concatenat
e,Up_Convolution])

        Convolution1 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv1", activation = 'r
elu', kernel_initializer= initialize, padding ='same')(Concatenated_Layer)
        Convolution2 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv2", activation = 'r
elu', kernel_initializer= initialize, padding ='same')(Convolution1)

        if Block_Number==4:

            Convolution3 = tf.keras.layers.Conv2D(n_classes, Kernel_Size, name= "Final_Conv", activation
= 'relu', kernel_initializer= initialize, padding ='same')(Convolution2)
            Output=Activation('softmax', name="Softmax")(Convolution3)

            return Output

        return Convolution2

    Input_layer = tf.keras.layers.Input(shape=input_shape)
    En_Block1 = Unet_En_Blocks(1, "En_Block1", 64,  (3,3), (2,2), Input_layer)
    En_Block2 = Unet_En_Blocks(2, "En_Block2", 128, (3,3), (2,2), En_Block1)
    En_Block3 = Unet_En_Blocks(3, "En_Block3", 256, (3,3), (2,2), En_Block2)
    En_Block4 = Unet_En_Blocks(4, "En_Block4", 512, (3,3), (2,2), En_Block3)
    En_Block5 = Unet_En_Blocks(5, "En_Block5", 1024, (3,3), (2,2), En_Block4)

    Dec_Block1 = Unet_Dec_Blocks(1, "Dec_Block1", 512,  (3,3), En_Block5, En_Block4)
    Dec_Block2 = Unet_Dec_Blocks(2, "Dec_Block2", 256, (3,3), Dec_Block1, En_Block3)
    Dec_Block3 = Unet_Dec_Blocks(3, "Dec_Block3", 128, (3,3), Dec_Block2, En_Block2)
    Output_layer = Unet_Dec_Blocks(4, "Dec_Block4", 64, (3,3), Dec_Block3, En_Block1)

    Unet_model = Model(Input_layer, Output_layer)

    return Unet_model
```

### Deeplab_V3 Definition for prediction

In [ ]:

```
def Deeplab_V3(Input_shape, n_classes):

    def Restnet_for_Deeplab(Input_layer):

        def Restnet_Conv_Block(Block_Number, Sub_block, Name, Filters, Previous_layer, initialize="he_nor
mal"):

            Multi_grid, Rate = [1,2,4], 2
            strides=(1,1) if Block_Number==2 or Block_Number==5 else (2,2)

            if (Block_Number==5) and (Sub_block==1):
                D_Rate=Multi_grid[0] * Rate
            elif (Block_Number==5) and (Sub_block==2):
                D_Rate=Multi_grid[1] * Rate
            elif (Block_Number==5) and (Sub_block==3):
                D_Rate=Multi_grid[2] * Rate
            else:D_Rate=1

            Convolution1 = tf.keras.layers.Conv2D(Filters[0],(1,1), strides=strides, name= Name+"Conv1",
activation = 'relu', kernel_initializer=initialize)(Previous_layer)
            Batch_norm1=tf.keras.layers.BatchNormalization()(Convolution1)

            Convolution2 = tf.keras.layers.Conv2D(Filters[1],(3,3), dilation_rate=D_Rate, name= Name+"Co
nv2", padding='same', activation = 'relu', kernel_initializer=initialize)(Batch_norm1)
            Batch_norm2=tf.keras.layers.BatchNormalization()(Convolution2)

            Convolution3 = tf.keras.layers.Conv2D(Filters[2],(1,1),name= Name+"Conv3", activation = None
, kernel_initializer=initialize)(Batch_norm2)
            Batch_norm3=tf.keras.layers.BatchNormalization()(Convolution3)

            Layer_to_add = tf.keras.layers.Conv2D(Filters[2],(1,1), strides=strides, name= Name+"conv_pr
ep_add", activation = None, kernel_initializer=initialize)(Previous_layer)

            Layer_to_add=tf.keras.layers.BatchNormalization()(Layer_to_add)

            Added_Layer=tf.keras.layers.add([Batch_norm3,Layer_to_add])
            Final_Conv=tf.keras.layers.Activation("relu")(Added_Layer)

            return Final_Conv

        def Restnet_Id_Block(Block_Number,Sub_block,Name,Filters,Previous_layer,initialize="he_normal"):
```

```
            Multi_grid, Rate = [1,2,4], 2
            if (Block_Number==5) and (Sub_block==1):
                D_Rate=Multi_grid[0] * Rate
            elif (Block_Number==5) and (Sub_block==2):
                D_Rate=Multi_grid[1] * Rate
            elif (Block_Number==5) and (Sub_block==3):
                D_Rate=Multi_grid[2] * Rate
            else:D_Rate=1

            Convolution1 = tf.keras.layers.Conv2D(Filters[0], (1,1), name= Name+"Conv1", activation = 'r
elu', kernel_initializer=initialize)(Previous_layer)
            Batch_norm1=tf.keras.layers.BatchNormalization()(Convolution1)

            Convolution2 = tf.keras.layers.Conv2D(Filters[1], (3,3), dilation_rate=D_Rate, name= Name+"C
onv2", padding='same', activation = 'relu', kernel_initializer=initialize)(Batch_norm1)
            Batch_norm2=tf.keras.layers.BatchNormalization()(Convolution2)

            Convolution3 = tf.keras.layers.Conv2D(Filters[2], (1,1), name= Name+"Conv3", activation = No
ne, kernel_initializer=initialize)(Batch_norm2)
            Batch_norm3=tf.keras.layers.BatchNormalization()(Convolution3)

            Added_Layer=tf.keras.layers.add([Batch_norm3,Previous_layer])
            Final_Conv1=tf.keras.layers.Activation("relu")(Added_Layer)

            return Final_Conv1

        Block_1_Conv = tf.keras.layers.Conv2D(64,(7,7),strides=(2,2),name= "Block1_Conv1", activation =
'relu', padding='same', kernel_initializer="he_normal")(Input_layer)
        Block_1_Batch_norm1=tf.keras.layers.BatchNormalization(name="Block1_Conv1_BN")(Block_1_Conv)
        Block_1_MaxPool = tf.keras.layers.MaxPooling2D(pool_size=(3,3),strides=(2,2), name="Block1_Maxpoo
l1", padding='same')(Block_1_Batch_norm1)

        Block2_1_con= Restnet_Conv_Block(2,1,"Block2.1_CONV_", [64,64,256], Block_1_MaxPool)
        Block2_2_ID= Restnet_Id_Block( 2, 2,"Block2.2_ID_",    [64,64,256], Block2_1_con)
        Block2_3_ID= Restnet_Id_Block( 2, 3,"Block2.3_ID_",    [64,64,256], Block2_2_ID)

        Block3_1_con= Restnet_Conv_Block(3, 1,"Block3.1_CONV_", [128,128,512], Block2_3_ID)
        Block3_2_ID= Restnet_Id_Block( 3,   2, "Block3.2_ID_",   [128,128,512], Block3_1_con)
        Block3_3_ID= Restnet_Id_Block( 3,   3, "Block3.3_ID_",   [128,128,512], Block3_2_ID)
        Block3_4_ID= Restnet_Id_Block( 3,   4, "Block3.4_ID_",   [128,128,512], Block3_3_ID)

        Block4_1_con= Restnet_Conv_Block(4, 1, "Block4.1_CONV_", [256,256,1024], Block3_4_ID)
        Block4_2_ID= Restnet_Id_Block( 4,   2, "Block4.2_ID_",   [256,256,1024], Block4_1_con)
        Block4_3_ID= Restnet_Id_Block( 4,   3, "Block4.3_ID_",   [256,256,1024], Block4_2_ID)
        Block4_4_ID= Restnet_Id_Block( 4,   4, "Block4.4_ID_",   [256,256,1024], Block4_3_ID)
        Block4_5_ID= Restnet_Id_Block( 4,   5, "Block4.5_ID_",   [256,256,1024], Block4_4_ID)
        Block4_6_ID= Restnet_Id_Block( 4,   6, "Block4.6_ID_",   [256,256,1024], Block4_5_ID)

        Block5_1_con= Restnet_Conv_Block(5, 1, "Block5.1_CONV_", [512,512,2048], Block4_6_ID)
        Block5_2_ID= Restnet_Id_Block( 5,   2, "Block5.2_ID_",   [512,512,2048], Block5_1_con)
        Block5_3_ID= Restnet_Id_Block( 5,   3, "Block5.3_ID_",   [512,512,2048], Block5_2_ID)

        return Block5_3_ID

    def Atrous_Spatial_Pyramid_Pooling(Restnet_Block,ASPP_Filter=256):

        G_pool = tf.keras.layers.GlobalAveragePooling2D(name='ASPP_GL_POOL')(Restnet_Block)
        G_pool = tf.keras.layers.Reshape((1,1,Restnet_Block.shape[3]))(G_pool)

        Convolution1 = tf.keras.layers.Conv2D(ASPP_Filter, (1,1), name= "Global_Conv", activation = "rel
u")(G_pool)
        Batch_norm1=tf.keras.layers.BatchNormalization()(Convolution1)

        Global_Features = UpSampling2D( size=(15,30), interpolation='bilinear', name='upsamp')(Batch_norm
1)

        ASPP_Conv0 = tf.keras.layers.Conv2D(ASPP_Filter, (1,1), name= "ASPP_Conv0_1x1", activation = "re
lu", padding="same")(Restnet_Block)
        ASPP_Conv0=tf.keras.layers.BatchNormalization()(ASPP_Conv0)

        ASPP_Conv1 = tf.keras.layers.Conv2D(ASPP_Filter, (3,3), name= "ASPP_Conv1_3x3", dilation_rate=6,
activation = "relu", padding="same")(Restnet_Block)
        ASPP_Conv1=tf.keras.layers.BatchNormalization()(ASPP_Conv1)

        ASPP_Conv2 = tf.keras.layers.Conv2D(ASPP_Filter, (3,3), name= "ASPP_Conv2_3x3", dilation_rate=12
, activation = "relu", padding="same")(Restnet_Block)
        ASPP_Conv2=tf.keras.layers.BatchNormalization()(ASPP_Conv2)

        ASPP_Conv3 = tf.keras.layers.Conv2D(ASPP_Filter, (3,3), name= "ASPP_Conv3_3x3", dilation_rate=18
, activation = "relu", padding="same")(Restnet_Block)
        ASPP_Conv3=tf.keras.layers.BatchNormalization()(ASPP_Conv3)

        ASPP_Features= tf.keras.layers.concatenate([Global_Features, ASPP_Conv0, ASPP_Conv1, ASPP_Conv2,
```

```
ASPP_Conv3])
        ASPP_Features = tf.keras.layers.Conv2D(ASPP_Filter, (1,1), name= "ASPP_Out", activation = "relu"
)(ASPP_Features)
        ASPP_Features=tf.keras.layers.BatchNormalization()(ASPP_Features)

        ASPP_Features = UpSampling2D(size=(16,16), interpolation='bilinear', name='upsampling')(ASPP_Feat
ures)

        ASPP_Features = tf.keras.layers.Conv2D(64, (1,1), name= "out", activation = "relu")(ASPP_Feature
s)

        ASPP_Features = tf.keras.layers.Conv2D(n_classes, (1,1), name= "output", activation ='relu')(ASP
P_Features)

        Output=Activation('softmax', name="Softmax")(ASPP_Features)

        return Output

    Input_layer = tf.keras.layers.Input(shape=Input_shape)
    Block5_3_ID= Restnet_for_Deeplab(Input_layer)
    Output=Atrous_Spatial_Pyramid_Pooling(Block5_3_ID)
    model = Model(inputs=Input_layer, outputs=Output, name="Deeplab_V3")

    return model
```

### Segnet Definition for prediction

In [ ]:

```
from keras.layers import *
from keras.models import *
import numpy as np

def MaxUnpooling2D(pool, ind, batch_size, name):
    with tf.compat.v1.variable_scope(name):
        output_shape=[None,pool.shape[1]*2,pool.shape[2]*2,pool.shape[3]]
        pool_ = tf.reshape(pool, [-1])
        batch_range = tf.reshape(tf.range(batch_size, dtype=ind.dtype), [tf.shape(pool)[0], 1, 1, 1])
        b = tf.ones_like(ind) * batch_range
        b = tf.reshape(b, [-1, 1])
        ind_ = tf.reshape(ind, [-1, 1])
        ind_ = tf.concat([b, ind_], 1)
        ret = tf.scatter_nd(ind_, pool_, shape=[batch_size, output_shape[1] * output_shape[2] * output_s
hape[3]])
        ret = tf.reshape(ret, [tf.shape(pool)[0], output_shape[1], output_shape[2], output_shape[3]])
        return ret

def Segnet_Segmentation(input_shape, n_labels, batch_size, Kernel=(3,3), Pool_Filter=(2,2), output_mode=
"softmax"):

  Inputs, batch_size = Input(shape=input_shape), batch_size

  def Segnet_Encoder(Block_Number, Filters, Input_layer):

    Layer_1 = Convolution2D(Filters, Kernel, name= "En_Block"+str(Block_Number)+"_Conv1", activation = 'r
elu', padding= "same",  kernel_initializer= "he_normal")(Input_layer)
    Layer_1 = BatchNormalization(name= "En_Block"+str(Block_Number)+"_Batch1")(Layer_1)

    Layer_2 = Convolution2D(Filters, Kernel, name= "En_Block"+str(Block_Number)+"_Conv2", activation = 'r
elu', padding= "same",  kernel_initializer= "he_normal")(Layer_1)
    Layer_2 = BatchNormalization(name= "En_Block"+str(Block_Number)+"_Batch2")(Layer_2)

    if Block_Number>2:
      Layer_3 = Convolution2D(Filters, Kernel, name= "En_Block"+str(Block_Number)+"_Conv3", activation =
'relu', padding= "same",  kernel_initializer= "he_normal")(Layer_2)
      Layer_3 = BatchNormalization(name= "En_Block"+str(Block_Number)+"_Batch3")(Layer_3)

      if Block_Number==5:
        Layer_3=ZeroPadding2D(((1,0),(0,0)), name='Zero_pad')(Layer_3)
      return Layer_3

    return Layer_2

  def Segnet_Decoder(Block_Number, Filters, Input_layer):

    Get_filter = lambda x : int(Filters/2) if (((Block_Number > 2 and Block_Number < 5) and (x==3)) or (
Block_Number==2 and x==2)) else Filters

    if Block_Number==5:
        Input_layer=Cropping2D(((1, 0),(0,0)))(Input_layer)
```

```
    Layer_1 = Convolution2D(Get_filter(1), Kernel, name= "Dec_Block"+str(Block_Number)+"_Conv1", activati
on = 'relu', padding= "same",  kernel_initializer= "he_normal")(Input_layer)
    Layer_1 = BatchNormalization(name= "Dec_Block"+str(Block_Number)+"_Batch1")(Layer_1)

    Layer_2 = Convolution2D(Get_filter(2), Kernel, name= "Dec_Block"+str(Block_Number)+"_Conv2", activati
on = 'relu', padding= "same",  kernel_initializer= "he_normal")(Layer_1)
    Layer_2 = BatchNormalization(name= "Dec_Block"+str(Block_Number)+"_Batch2")(Layer_2)

    if Block_Number>2:
      Layer_3 = Convolution2D(Get_filter(3), Kernel, name= "Dec_Block"+str(Block_Number)+"_Conv3", activ
ation = 'relu', padding= "same",  kernel_initializer= "he_normal")(Layer_2)
      Layer_3 = BatchNormalization(name= "Dec_Block"+str(Block_Number)+"_Batch3")(Layer_3)
      return Layer_3
    return Layer_2

  Encoder_Conv1=Segnet_Encoder(1, 64, Inputs)
  max_pool1,pool_indices_1 =tf.nn.max_pool_with_argmax(Encoder_Conv1, [1, 2, 2, 1], [1, 2, 2, 1], paddin
g="VALID")

  Encoder_Conv2=Segnet_Encoder(2, 128, max_pool1)
  max_pool_2,pool_indices_2 =tf.nn.max_pool_with_argmax(Encoder_Conv2, [1, 2, 2, 1], [1, 2, 2, 1], paddi
ng="VALID")

  Encoder_Conv3=Segnet_Encoder(3, 256, max_pool_2)
  max_pool_3,pool_indices_3 =tf.nn.max_pool_with_argmax(Encoder_Conv3, [1, 2, 2, 1], [1, 2, 2, 1], paddi
ng="VALID")

  Encoder_Conv4=Segnet_Encoder(4, 512, max_pool_3)
  max_pool_4,pool_indices_4 =tf.nn.max_pool_with_argmax(Encoder_Conv4, [1, 2, 2, 1], [1, 2, 2, 1], paddi
ng="VALID")

  Encoder_Conv5=Segnet_Encoder(5, 512, max_pool_4)
  max_pool_5,pool_indices_5 =tf.nn.max_pool_with_argmax(Encoder_Conv5, [1, 2, 2, 1], [1, 2, 2, 1], paddi
ng="VALID")

  Decoder_upsamp5 =MaxUnpooling2D(max_pool_5, pool_indices_5,  batch_size, name="un_pool_5")
  Decoder_Conv5=Segnet_Decoder(5, 512, Decoder_upsamp5)

  Decoder_upsamp4 =MaxUnpooling2D(Decoder_Conv5, pool_indices_4, batch_size, name="un_pool_4")
  Decoder_Conv4=Segnet_Decoder(4, 512, Decoder_upsamp4)

  Decoder_upsamp3 =MaxUnpooling2D(Decoder_Conv4, pool_indices_3, batch_size, name="un_pool_3")
  Decoder_Conv3=Segnet_Decoder(3, 256, Decoder_upsamp3)

  Decoder_upsamp2 =MaxUnpooling2D(Decoder_Conv3, pool_indices_2, batch_size, name="un_pool_2")
  Decoder_Conv2=Segnet_Decoder(2, 128, Decoder_upsamp2)

  Decoder_upsamp1 =MaxUnpooling2D(Decoder_Conv2, pool_indices_1, batch_size, name="un_pool_1")
  Decoder_Conv1=Segnet_Decoder(1, 64, Decoder_upsamp1)

  Convolution_out = tf.keras.layers.Conv2D(n_labels, (3,3), name= "Final_Conv", activation = 'relu', ker
nel_initializer="he_normal", padding ='same')(Decoder_Conv1)
  Output=Activation('softmax', name="Softmax")(Convolution_out)

  model = Model(inputs=Inputs, outputs=Output, name="SEGNET")

  return model
```

### PSPNET Definition for prediction

In [ ]:

```
def PSPNET(Input_shape, n_classes):

    def Restnet50_Module(Input_layer, n_classes):

        def Restnet_Conv_Block(Block_Number,Name,Filters,Previous_layer,initialize="he_normal"):

            strides=(1,1) if Block_Number==2 else (2,2)

            if (Block_Number==4):
                D_Rate=2
            elif (Block_Number==5):
                D_Rate=4
            else:
                D_Rate=1

            Convolution1 = tf.keras.layers.Conv2D(Filters[0],(1,1), strides=strides, name= Name+"Conv1",
activation = 'relu', kernel_initializer=initialize)(Previous_layer)
            Batch_norm1=tf.keras.layers.BatchNormalization()(Convolution1)
```

```
            Convolution2 = tf.keras.layers.Conv2D(Filters[1],(3,3), dilation_rate=D_Rate, name= Name+"Co
nv2", padding='same', activation = 'relu', kernel_initializer=initialize)(Batch_norm1)
            Batch_norm2=tf.keras.layers.BatchNormalization()(Convolution2)

            Convolution3 = tf.keras.layers.Conv2D(Filters[2],(1,1),name= Name+"Conv3", activation = None
, kernel_initializer=initialize)(Batch_norm2)
            Batch_norm3=tf.keras.layers.BatchNormalization()(Convolution3)

            Layer_to_add = tf.keras.layers.Conv2D(Filters[2],(1,1), strides=strides, name= Name+"conv_pr
ep_add", activation = None, kernel_initializer=initialize)(Previous_layer)

            Layer_to_add=tf.keras.layers.BatchNormalization()(Layer_to_add)

            Added_Layer=tf.keras.layers.add([Batch_norm3,Layer_to_add])
            Final_Conv=tf.keras.layers.Activation("relu")(Added_Layer)

            return Final_Conv

        def Restnet_Id_Block(Block_Number,Name,Filters,Previous_layer,initialize="he_normal"):

            if (Block_Number==4):
                D_Rate=2
            elif (Block_Number==5):
                D_Rate=4
            else:
                D_Rate=1

            Convolution1 = tf.keras.layers.Conv2D(Filters[0], (1,1), name= Name+"Conv1", activation = 'r
elu', kernel_initializer=initialize)(Previous_layer)
            Batch_norm1=tf.keras.layers.BatchNormalization()(Convolution1)

            Convolution2 = tf.keras.layers.Conv2D(Filters[1], (3,3), dilation_rate=D_Rate, name= Name+"C
onv2", padding='same', activation = 'relu', kernel_initializer=initialize)(Batch_norm1)
            Batch_norm2=tf.keras.layers.BatchNormalization()(Convolution2)

            Convolution3 = tf.keras.layers.Conv2D(Filters[2], (1,1), name= Name+"Conv3", activation = No
ne, kernel_initializer=initialize)(Batch_norm2)
            Batch_norm3=tf.keras.layers.BatchNormalization()(Convolution3)

            Added_Layer=tf.keras.layers.add([Batch_norm3,Previous_layer])
            Final_Conv1=tf.keras.layers.Activation("relu")(Added_Layer)

            return Final_Conv1

        Block_1_Conv = tf.keras.layers.Conv2D(64,(7,7),strides=(2,2),name= "Block1_Conv1", activation =
'relu', kernel_initializer="he_normal")(Input_layer)
        Block_1_Batch_norm1=tf.keras.layers.BatchNormalization(name="Block1_Conv1_BN")(Block_1_Conv)
        Block_1_MaxPool = tf.keras.layers.MaxPooling2D(pool_size=(3,3),strides=(2,2), name="Block1_Maxpoo
l1")(Block_1_Batch_norm1)

        Block2_1_con= Restnet_Conv_Block(2, "Block2.1_CONV_", [16,16,64], Block_1_MaxPool)
        Block2_2_ID= Restnet_Id_Block( 2,  "Block2.2_ID_",    [16,16,64], Block2_1_con)
        Block2_3_ID= Restnet_Id_Block( 2,  "Block2.3_ID_",    [16,16,64], Block2_2_ID)


        Block3_1_con= Restnet_Conv_Block(3, "Block3.1_CONV_", [32,32,128], Block2_3_ID)
        Block3_2_ID= Restnet_Id_Block( 3,   "Block3.2_ID_",   [32,32,128], Block3_1_con)
        Block3_3_ID= Restnet_Id_Block( 3,   "Block3.3_ID_",   [32,32,128], Block3_2_ID)
        Block3_4_ID= Restnet_Id_Block( 3,   "Block3.4_ID_",   [32,32,128], Block3_3_ID)

        Block4_1_con= Restnet_Conv_Block(4, "Block4.1_CONV_", [64,64,256], Block3_4_ID)
        Block4_2_ID= Restnet_Id_Block( 4,   "Block4.2_ID_",   [64,64,256], Block4_1_con)
        Block4_3_ID= Restnet_Id_Block( 4,   "Block4.3_ID_",   [64,64,256], Block4_2_ID)
        Block4_4_ID= Restnet_Id_Block( 4,   "Block4.4_ID_",   [64,64,256], Block4_3_ID)
        Block4_5_ID= Restnet_Id_Block( 4,   "Block4.5_ID_",   [64,64,256], Block4_4_ID)
        Block4_6_ID= Restnet_Id_Block( 4,   "Block4.6_ID_",   [64,64,256], Block4_5_ID)

        Block5_1_con= Restnet_Conv_Block(5, "Block5.1_CONV_", [128,128,512], Block4_6_ID)
        Block5_2_ID= Restnet_Id_Block( 5,   "Block5.2_ID_",   [128,128,512], Block5_1_con)
        Block5_3_ID= Restnet_Id_Block( 5,   "Block5.3_ID_",   [128,128,512], Block5_2_ID)

        return Block5_3_ID

    def Pyramid_Module(Rest50_Layer):

        def Feature_Sub_Map(Sub_block, Pool_size, Previous_layer, filters=128):

            if Sub_block=="RED":
                Pool_size= (Previous_layer.shape[1],Previous_layer.shape[2])
                Sub_pool = tf.keras.layers.GlobalAveragePooling2D(name= Sub_block+'_GL_POOL')(Previous_l
ayer)
                Sub_pool = tf.keras.layers.Reshape((1,1,Previous_layer.shape[3]))(Sub_pool)
            else:
```

```
                Sub_pool = tf.keras.layers.AveragePooling2D(pool_size=Pool_size,name= Sub_block+'_AVG_PO
OL')(Previous_layer)

            Conv_sub = tf.keras.layers.Convolution2D(filters=filters,kernel_size=(1,1),name= Sub_block+'
Conv1_1')(Sub_pool)
            Conv_sub=tf.keras.layers.BatchNormalization()(Conv_sub)
            Retn_Sub = tf.keras.layers.UpSampling2D(size=Pool_size, name=Sub_block+'Up_sample',interpolat
ion='bilinear')(Conv_sub)

            return Retn_Sub

        Rest50_Layer = UpSampling2D(size=(4,4),interpolation='bilinear',name='Size_Adjust_samp')(Rest50_L
ayer)
        Rest50_Layer=tf.keras.layers.ZeroPadding2D((2,0),name='up_Size_Adjust_pad')(Rest50_Layer)

        Red_Map= Feature_Sub_Map(    "RED",    (1,1), Rest50_Layer)
        Orange_Map= Feature_Sub_Map("ORANGE", (2,2), Rest50_Layer)
        Blue_Map= Feature_Sub_Map(   "BLUE",   (3,3), Rest50_Layer)
        Green_Map= Feature_Sub_Map(  "GREEN",  (6,6), Rest50_Layer)

        Global_Concat= tf.keras.layers.concatenate([Rest50_Layer, Green_Map, Blue_Map, Orange_Map, Red_Ma
p])
        UpSampling = tf.keras.layers.UpSampling2D(size=(6,8),interpolation='bilinear',name='en_Size_Adjus
t_samp')(Global_Concat)

        Convolution1 = tf.keras.layers.Conv2D(64, (5,5),name= "Conv1", activation = 'relu', padding="sam
e",kernel_initializer="he_normal")(ZeroPadding2D((12,0))(UpSampling))
        Convolution1=tf.keras.layers.BatchNormalization()(Convolution1)

        Convolution2 = tf.keras.layers.Conv2D(7, (3,3),name= "Conv2",activation = 'relu', padding="same"
,kernel_initializer="he_normal")(Convolution1)
        Output=Activation('softmax', name="Softmax")(Convolution2)

        return Output

    Input_layer = tf.keras.layers.Input(shape=Input_shape)
    Rest50_Layer= Restnet50_Module(Input_layer, n_classes)
    Output=Pyramid_Module(Rest50_Layer)
    PSPNET_Model = Model(Input_layer, Output)

    return PSPNET_Model
```