Segmentation of Indian Traffic

In [4]:
```
import math
from PIL import Image, ImageDraw
from PIL import ImagePath
import pandas as pd
import os
from os import path
from tqdm import tqdm
import json
import cv2
import numpy as np
import matplotlib.pyplot as plt
import urllib
import tensorflow as tf
# tf.enable_eager_execution()
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
# from hilbert import hilbertCurve
import imgaug.augmenters as iaa
import numpy as np
# import albumentations as A
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Activation,Dropout,Flatten,BatchNormalization, ReLU, Reshape
#from tensorflow.keras.layers import Dense,Input,Conv2D,MaxPool2D,Activation,Dropout,Flatten, CuDNNLSTM, BatchNormalization, ReLU, Reshape
from tensorflow.keras.models import Model
import random as rn
```

1. You can download the data from this link, and extract it

2. All your data will be in the folder "data"

3. Inside the data you will be having two folders

```
|--- data
|-----| ---- images
|-----| ------|----- Scene 1
|-----| ------|--------| ----- Frame 1 (image 1)
|-----| ------|--------| ----- Frame 2 (image 2)
|-----| ------|--------| ----- ...
|-----| ------|----- Scene 2
|-----| ------|--------| ----- Frame 1 (image 1)
|-----| ------|--------| ----- Frame 2 (image 2)
|-----| ------|--------| ----- ...
|-----| ------|----- .....
|-----| ---- masks
|-----| ------|----- Scene 1
|-----| ------|--------| ----- json 1 (labeled objects in image 1)
|-----| ------|--------| ----- json 2 (labeled objects in image 1)
|-----| ------|--------| ----- ...
|-----| ------|----- Scene 2
|-----| ------|--------| ----- json 1 (labeled objects in image 1)
|-----| ------|--------| ----- json 2 (labeled objects in image 1)
|-----| ------|--------| ----- ...
|-----| ------|----- .....
```

Task 1: Preprocessing

1. Get all the file name and corresponding json files

In [ ]:
```
def return_file_names_df(root_dir):
  paths = [list(),list()]
  data_files = os.listdir(root_dir)
  for i,sub_file in enumerate(data_files):
    sub_file_path = root_dir + '/' + sub_file + '/'
    folder = os.listdir(sub_file_path)
    for folder_name in folder:
      path_name_folder_name = root_dir + '/' + sub_file + '/' + folder_name + '/'
      files = os.listdir(path_name_folder_name)
      for file in files:
        if (file.endswith("jpg")):
          path_name = root_dir + '/' + sub_file + '/' + folder_name + '/' + file
          path_name = str(path_name)
          paths[i].append(path_name)

        if (file.endswith("json")):
          path_name = root_dir + '/' + sub_file + '/' + folder_name + '/' + file
```

```
            path_name = str(path_name)
            paths[i].append(path_name)

    # intialise data of lists.
    data = {'image':paths[0],
            'json':paths[1]}

    # Create DataFrame
    data_df = pd.DataFrame(data)

    return data_df
```

In [ ]:
```
root_dir = 'data'
data_df = return_file_names_df(root_dir)
data_df.head()
```

Out[ ]:

	image	json
0	data/images/201/frame0029_leftImg8bit.jpg	data/mask/201/frame0029_gtFine_polygons.json
1	data/images/201/frame0299_leftImg8bit.jpg	data/mask/201/frame0299_gtFine_polygons.json
2	data/images/201/frame0779_leftImg8bit.jpg	data/mask/201/frame0779_gtFine_polygons.json
3	data/images/201/frame1019_leftImg8bit.jpg	data/mask/201/frame1019_gtFine_polygons.json
4	data/images/201/frame1469_leftImg8bit.jpg	data/mask/201/frame1469_gtFine_polygons.json

In [ ]:
```
import json

# Opening JSON file
f = open('data/mask/201/frame0029_gtFine_polygons.json',)

# returns JSON object as
# a dictionary
data = json.load(f)

# Iterating through the json
# list
labels = []
for i in data['objects']:
  #print(i['label'])
  labels.append(i['label'])


# Closing file
f.close()
```

In [ ]:
```
print(len(list(set(labels))))
```

```
18
```

> If you observe the dataframe, we can consider each row as single data point, where first feature is image and the second feature is corresponding json file

In [ ]:
```
def grader_1(data_df):
    for i in data_df.values:
        if not (path.isfile(i[0]) and path.isfile(i[1]) and i[0][12:i[0].find('_')]==i[1][10:i[1].find('_')]):
            return False
    return True
```

In [ ]:
```
grader_1(data_df)
```

Out[ ]:
```
True
```

2. Structure of sample Json file

```
"imgHeight": 1080,
"imgWidth": 1920,
"objects": [
    {
        "date": "25-Jun-2019 23:13:12",
        "deleted": 0,
        "draw": true,
        "id": 0,
        "label": "sky",
        "polygon": [
            [
                0.0,
                556.1538461538462
            ],
            [
                810.0,
                565.3846153846154
            ],
            [
                1374.2307692307693,
                596.5384615384615
            ],
            [
                1919.0,
                639.2307692307692
            ],
            [
                1919.0,
                0.0
            ],
            [
                0.0,
                0.0
            ]
        ],
        "user": "cvit",
        "verified": 0
    },
```


- Each File will have 3 attributes
  - imgHeight: which tells the height of the image
  - imgWidth: which tells the width of the image
  - objects: it is a list of objects, each object will have multiple attributes,
    - label: the type of the object
    - polygon: a list of two element lists, representing the coordinates of the polygon

Compute the unique labels

Let's see how many unique objects are there in the json file. to see how to get the object from the json file please check this blog

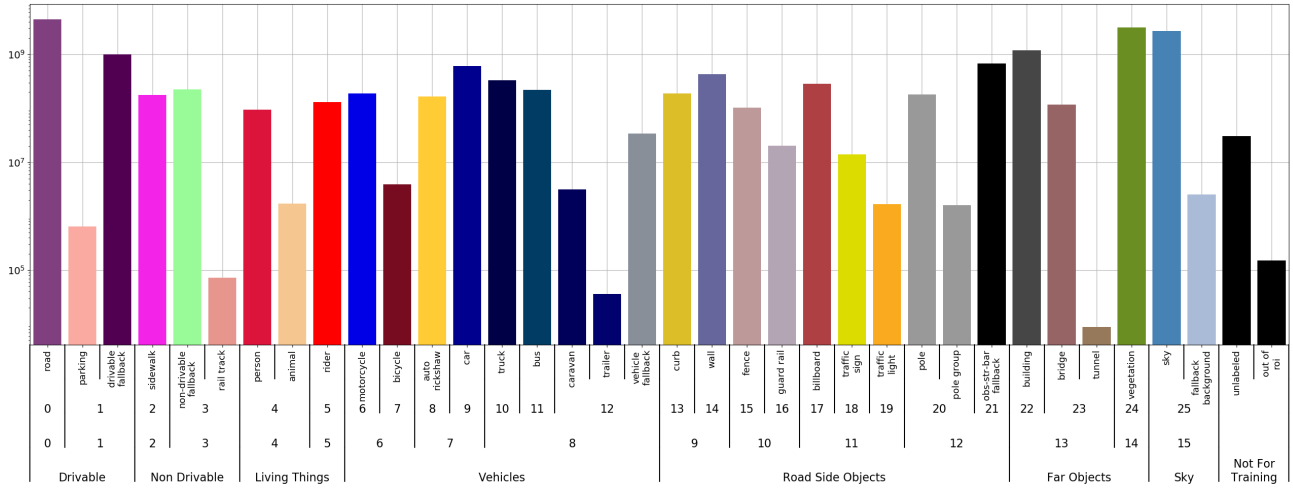
In [ ]:
```
def return_unique_labels(data_df):
  unique_labels = dict()
  for i , row in data_df.iterrows():
    # Opening JSON file
    f = open(row['json'],)
    data = json.load(f)
    labels = []
    for i in data['objects']:
      labels.append(i['label'])

    labels = list(set(labels))

    for label in labels:
      unique_labels[label] = unique_labels.get(label, 0) + 1
      #if label not in unique_labels:
        #unique_labels[label] += 1
  return unique_labels
```

In [ ]:
```
unique_labels = return_unique_labels(data_df)
```

In [ ]:
```
unique_labels
```

Out[ ]:
```
{'animal': 341,
 'autorickshaw': 2035,
 'bicycle': 321,
 'billboard': 2509,
 'bridge': 387,
 'building': 3136,
 'bus': 914,
 'car': 2981,
 'caravan': 53,
 'curb': 2447,
 'drivable fallback': 3616,
 'ego vehicle': 2,
 'fallback background': 893,
 'fence': 1024,
 'ground': 2,
 'guard rail': 728,
 'motorcycle': 2955,
 'non-drivable fallback': 2931,
 'obs-str-bar-fallback': 3972,
 'out of roi': 127,
 'parking': 17,
 'person': 2422,
 'pole': 3866,
 'polegroup': 337,
 'rail track': 4,
 'rectification border': 3,
 'rider': 2825,
 'road': 3861,
 'sidewalk': 617,
 'sky': 3991,
```

```
'traffic light': 167,
'traffic sign': 1728,
'trailer': 6,
'train': 4,
'truck': 2090,
'tunnel': 5,
'unlabeled': 2,
'vegetation': 3984,
'vehicle fallback': 2459,
'wall': 2180}
```



In [5]:
```
label_clr = {'road':10, 'parking':20, 'drivable fallback':20,'sidewalk':30,'non-drivable fallback':40,'rail track':40,\
                        'person':50, 'animal':50, 'rider':60, 'motorcycle':70, 'bicycle':70, 'autorickshaw':80,\
                        'car':80, 'truck':90, 'bus':90, 'vehicle fallback':90, 'trailer':90, 'caravan':90,\
                        'curb':100, 'wall':100, 'fence':110,'guard rail':110, 'billboard':120,'traffic sign':120,\
                        'traffic light':120, 'pole':130, 'polegroup':130, 'obs-str-bar-fallback':130,'building':140,\
                        'bridge':140,'tunnel':140, 'vegetation':150, 'sky':160, 'fallback background':160,'unlabeled':0,\
                        'out of roi':0, 'ego vehicle':170, 'ground':180,'rectification border':190,\
                    'train':200}
```

In [6]:
```
class_label = [i for i in label_clr.values()]
class_label = list(set(class_label))
class_label = sorted(class_label)
class_label
```

Out[6]:
```
[0,
 10,
 20,
 30,
 40,
 50,
 60,
 70,
 80,
 90,
 100,
 110,
 120,
 130,
 140,
 150,
 160,
 170,
 180,
 190,
 200]
```

In [7]:
```
def grader_2(unique_labels):
    if (not (set(label_clr.keys())-set(unique_labels))) and len(unique_labels) == 40:
        print("True")
    else:
        print("Flase")

grader_2(label_clr)
```

```
True
```

* here we have given a number for each of object types, if you see we are having 21 different set of objects
* Note that we have multiplies each object's number with 10, that is just to make different objects look differently in the segmentation map
* Before you pass it to the models, you might need to devide the image array /10.

3. Extracting the polygons from the json files

In [ ]:
```
file = 'data/mask/201/frame0029_gtFine_polygons.json'
def get_poly(file):
  f = open(file,)
```

```
  data = json.load(f)
  h = data['imgHeight']
  w = data['imgWidth']
  label = []
  vertexlist = []
  for i in data['objects']:
    label.append(i['label'])
    polygon = i['polygon']
    polygon = [tuple(i) for i in polygon]
    vertexlist.append(polygon)


  return w, h, label, vertexlist
```

In [ ]:
```
def grader_3(file):
    w, h, labels, vertexlist = get_poly(file)
    print(len((set(labels)))==18 and len(vertexlist)==227 and w==1920 and h==1080 \
          and isinstance(vertexlist,list) and isinstance(vertexlist[0],list) and isinstance(vertexlist[0][0],tuple) )

grader_3('data/mask/201/frame0029_gtFine_polygons.json')
```

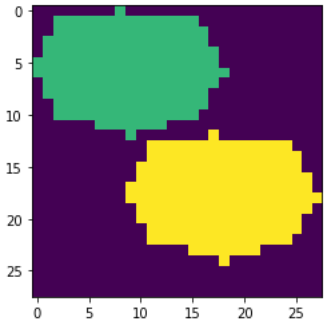
```
True
```

4. Creating Image segmentations by drawing set of polygons

Example

In [ ]:
```
import math
from PIL import Image, ImageDraw
from PIL import ImagePath
side=8
x1 = [ ((math.cos(th) + 1) *9, (math.sin(th) + 1) * 6) for th in [i * (2 * math.pi) / side for i in range(side)] ]
x2 = [ ((math.cos(th) + 2) *9, (math.sin(th) + 3) *6) for th in [i * (2 * math.pi) / side for i in range(side)] ]

img = Image.new("RGB", (28,28))
img1 = ImageDraw.Draw(img)
# please play with the fill value
# writing the first polygon
img1.polygon(x1, fill =20)
# writing the second polygon
img1.polygon(x2, fill =30)

img=np.array(img)
# note that the filling of the values happens at the channel 1, so we are considering only the first channel here
plt.imshow(img[:,:,0])
print(img.shape)
print(img[:,:,0]//10)
im = Image.fromarray(img[:,:,0])
im.save("test_image.png")
```

```
(28, 28, 3)
[[0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0]
 [0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
 [0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0]
 [0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0]
 [0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```



In [ ]:
```
file = 'data/mask/201/frame0029_gtFine_polygons.json'
w, h, labels, vertexlist = get_poly(file)
```

In [ ]:
```
def compute_masks(data_df):
  mask = []
  for i , row in data_df.iterrows():
    file = row['json']
    file1 = row['image']
    w, h, labels, vertexlist = get_poly(file)

    s = file1
    s = s.split('/')
    s = [i.split('.') for i in s]

    path = s[0][0] + '/' + 'output' + '/' + s[2][0] + '/' +s[3][0] + '.jpg'
    path = str(path)

    img = Image.new("RGB", (w, h))
    img1 = ImageDraw.Draw(img)
    found = 0
    for i in range(len(vertexlist)):
      if len(vertexlist[i]) == 0:
        found = 1

    if found == 0:
      for i in range(len(vertexlist)):
        img1.polygon(vertexlist[i], fill = label_clr[labels[i]])
      img=np.array(img)
      im = Image.fromarray(img[:,:,0])
      mask.append(path)
      im.save(path)
    else:
      mask.append('Nan')

  data_df['mask'] = mask

  return data_df
```

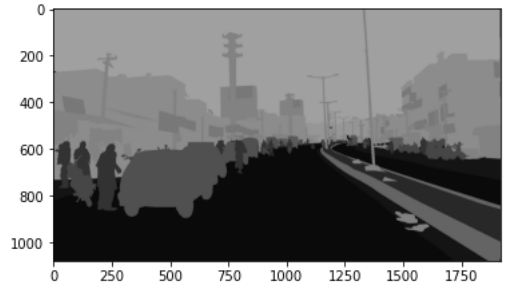
In [ ]:
```
data_df = compute_masks(data_df)
```

In [ ]:
```
data_df.to_csv('preprocessed_data.csv', index=False)
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
# i  created two dataframe one with .png format and another with .jpg file because .png file is not
# supported in further down data pipeline.
import urllib.request
def grader_3():
    url = "https://i.imgur.com/4XSUlHk.png"
    url_response = urllib.request.urlopen(url)
    img_array = np.array(bytearray(url_response.read()), dtype=np.uint8)
    img = cv2.imdecode(img_array, -1)
    my_img = cv2.imread('data4/output/201/frame0029_gtFine_polygons.png')
    plt.imshow(my_img)
    print((my_img[:,:,0]==img).all())
    print(np.unique(img))
    print(np.unique(my_img[:,:,0]))
    data_df.to_csv('preprocessed_data.csv', index=False)
grader_3()
```

```
True
[  0  10  20  40  50  60  70  80  90 100 120 130 140 150 160]
[  0  10  20  40  50  60  70  80  90 100 120 130 140 150 160]
```



In [ ]:
```
import pandas as pd
df = pd.read_csv('preprocessed_data1.csv')
```

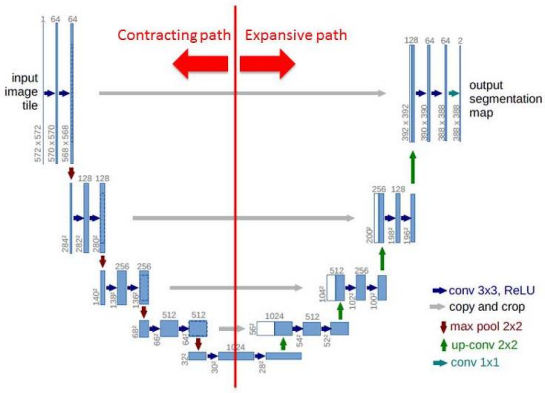
In [ ]:
```
d = df[(df['mask'] != 'Nan')]
d.shape
```

Out[ ]: (4001, 3)

Task 2: Applying Unet to segment the images

* please check the paper: https://arxiv.org/abs/1505.04597

Network Architecture



*

* As a part of this assignment we won't writingt this whole architecture, rather we will be doing transfer learning

* please check the library https://github.com/qubvel/segmentation_models

* You can install it like this "pip install -U segmentation-models==0.2.1", even in google colab you can install the same with "!pip install -U segmentation-models==0.2.1"

* Check the reference notebook in which we have solved one end to end case study of image forgery detection using same unet

* The number of channels in the output will depend on the number of classes in your data, since we know that we are having 21 classes, the number of channels in the output will also be 21

* **This is where we want you to explore, how do you featurize your created segmentation map note that the original map will be of (w, h, 1) and the output will be (w, h, 21) how will you calculate the loss**, you can check the examples in segmentation github

* please use the loss function that is used in the refence notebooks

In [ ]:
```
file_name = []
for i , row in d.iterrows():
  f = []
  s = row['image']
  t = row['mask']
  #s = s.split('/')
  #s = [i.split('.') for i in s]
  f.append(s)
  f.append(t)
  file_name.append(f)
```

In [ ]:
```
len(file_name)
```

Out[ ]:
```
4001
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(list(file_name), test_size=0.15, random_state=42)
```

In [ ]:
```
!pip install -U --pre segmentation-models
```

```
Collecting segmentation-models
  Downloading https://files.pythonhosted.org/packages/da/b9/4a183518c21689a56b834eaaa45cad242d9ec09a4360b5b10139f23c63f4/segmentation_models-1.0.1-py3-none-any.whl
Collecting efficientnet==1.0.0
  Downloading https://files.pythonhosted.org/packages/97/82/f3ae07316f0461417dc54affab6e86ab188a5a22f33176d35271628b96e0/efficientnet-1.0.0-py3-none-any.whl
Collecting image-classifiers==1.0.0
  Downloading https://files.pythonhosted.org/packages/81/98/6f84720e299a4942ab80df5f76ab97b7828b24d1de5e9b2cbbe6073228b7/image_classifiers-1.0.0-py3-none-any.whl
Collecting keras-applications<=1.0.8,>=1.0.7
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Applications-1.0.8-py3-none-any.whl (50kB)
     |████████████████████████████████| 51kB 4.3MB/s
Requirement already satisfied, skipping upgrade: scikit-image in /usr/local/lib/python3.7/dist-packages (from efficientnet==1.0.0->segmentation-models) (0.16.2)
Requirement already satisfied, skipping upgrade: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (from keras-applications<=1.0.8,>=1.0.7->segmentation-models) (1.19.5)
Requirement already satisfied, skipping upgrade: h5py in /usr/local/lib/python3.7/dist-packages (from keras-applications<=1.0.8,>=1.0.7->segmentation-models) (2.10.0)
Requirement already satisfied, skipping upgrade: PyWavelets>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (1.1.1)
Requirement already satisfied, skipping upgrade: matplotlib!=3.0.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (3.2.2)
Requirement already satisfied, skipping upgrade: pillow>=4.3.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (7.0.0)
Requirement already satisfied, skipping upgrade: scipy>=0.19.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (1.4.1)
Requirement already satisfied, skipping upgrade: networkx>=2.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (2.5)
Requirement already satisfied, skipping upgrade: imageio>=2.3.0 in /usr/local/lib/python3.7/dist-packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (2.4.1)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.7/dist-packages (from h5py->keras-applications<=1.0.8,>=1.0.7->segmentation-models) (1.15.0)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image->efficientnet==1.0.0->segmentation-models) (2.8.1)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image->efficientnet==1.0.0->segmentation-models) (1.3.1)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image->efficientnet==1.0.0->segmentation-models) (0.10.0)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.0->scikit-image->efficientnet==1.0.0->segmentation-models) (2.4.7)
Requirement already satisfied, skipping upgrade: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-packages (from networkx>=2.0->scikit-image->efficientnet==1.0.0->segmentation-models) (4.4.2)
Installing collected packages: keras-applications, efficientnet, image-classifiers, segmentation-models
Successfully installed efficientnet-1.0.0 image-classifiers-1.0.0 keras-applications-1.0.8 segmentation-models-1.0.1
```

In [ ]:
```
%env SM_FRAMEWORK=tf.keras
import segmentation_models as sm
from segmentation_models import Unet
sm.set_framework('tf.keras')
tf.keras.backend.set_image_data_format('channels_last')
```

```
env: SM_FRAMEWORK=tf.keras
Segmentation Models: using `tf.keras` framework.
```

In [ ]:
```
# loading the unet model and using the resnet 34 and initilized weights with imagenet weights
# "classes" :different types of classes in the dataset
model = Unet('resnet34', encoder_weights='imagenet', classes=21, activation='softmax', encoder_freeze = True, input_shape=(512,512,3))
```

```
Downloading data from https://github.com/qubvel/classification_models/releases/download/0.0.1/resnet34_imagenet_1000_no_top.h5
85524480/85521592 [==============================] - 1s 0us/step
```

In [ ]:
```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
data (InputLayer)               [(None, 512, 512, 3) 0
__________________________________________________________________________________________________
bn_data (BatchNormalization)    (None, 512, 512, 3)  9           data[0][0]
__________________________________________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 518, 518, 3)  0           bn_data[0][0]
__________________________________________________________________________________________________
conv0 (Conv2D)                  (None, 256, 256, 64) 9408        zero_padding2d[0][0]
__________________________________________________________________________________________________
bn0 (BatchNormalization)        (None, 256, 256, 64) 256         conv0[0][0]
__________________________________________________________________________________________________
relu0 (Activation)              (None, 256, 256, 64) 0           bn0[0][0]
__________________________________________________________________________________________________
zero_padding2d_1 (ZeroPadding2D (None, 258, 258, 64) 0           relu0[0][0]
__________________________________________________________________________________________________
pooling0 (MaxPooling2D)         (None, 128, 128, 64) 0           zero_padding2d_1[0][0]
__________________________________________________________________________________________________
stage1_unit1_bn1 (BatchNormaliz (None, 128, 128, 64) 256         pooling0[0][0]
__________________________________________________________________________________________________
stage1_unit1_relu1 (Activation) (None, 128, 128, 64) 0           stage1_unit1_bn1[0][0]
__________________________________________________________________________________________________
zero_padding2d_2 (ZeroPadding2D (None, 130, 130, 64) 0           stage1_unit1_relu1[0][0]
```

stage1_unit1_conv1 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_2[0][0]
stage1_unit1_bn2 (BatchNormaliz	(None, 128, 128, 64)	256	stage1_unit1_conv1[0][0]
stage1_unit1_relu2 (Activation)	(None, 128, 128, 64)	0	stage1_unit1_bn2[0][0]
zero_padding2d_3 (ZeroPadding2D	(None, 130, 130, 64)	0	stage1_unit1_relu2[0][0]
stage1_unit1_conv2 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_3[0][0]
stage1_unit1_sc (Conv2D)	(None, 128, 128, 64)	4096	stage1_unit1_relu1[0][0]
add (Add)	(None, 128, 128, 64)	0	stage1_unit1_conv2[0][0] stage1_unit1_sc[0][0]
stage1_unit2_bn1 (BatchNormaliz	(None, 128, 128, 64)	256	add[0][0]
stage1_unit2_relu1 (Activation)	(None, 128, 128, 64)	0	stage1_unit2_bn1[0][0]
zero_padding2d_4 (ZeroPadding2D	(None, 130, 130, 64)	0	stage1_unit2_relu1[0][0]
stage1_unit2_conv1 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_4[0][0]
stage1_unit2_bn2 (BatchNormaliz	(None, 128, 128, 64)	256	stage1_unit2_conv1[0][0]
stage1_unit2_relu2 (Activation)	(None, 128, 128, 64)	0	stage1_unit2_bn2[0][0]
zero_padding2d_5 (ZeroPadding2D	(None, 130, 130, 64)	0	stage1_unit2_relu2[0][0]
stage1_unit2_conv2 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_5[0][0]
add_1 (Add)	(None, 128, 128, 64)	0	stage1_unit2_conv2[0][0] add[0][0]
stage1_unit3_bn1 (BatchNormaliz	(None, 128, 128, 64)	256	add_1[0][0]
stage1_unit3_relu1 (Activation)	(None, 128, 128, 64)	0	stage1_unit3_bn1[0][0]
zero_padding2d_6 (ZeroPadding2D	(None, 130, 130, 64)	0	stage1_unit3_relu1[0][0]
stage1_unit3_conv1 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_6[0][0]
stage1_unit3_bn2 (BatchNormaliz	(None, 128, 128, 64)	256	stage1_unit3_conv1[0][0]
stage1_unit3_relu2 (Activation)	(None, 128, 128, 64)	0	stage1_unit3_bn2[0][0]
zero_padding2d_7 (ZeroPadding2D	(None, 130, 130, 64)	0	stage1_unit3_relu2[0][0]
stage1_unit3_conv2 (Conv2D)	(None, 128, 128, 64)	36864	zero_padding2d_7[0][0]
add_2 (Add)	(None, 128, 128, 64)	0	stage1_unit3_conv2[0][0] add_1[0][0]
stage2_unit1_bn1 (BatchNormaliz	(None, 128, 128, 64)	256	add_2[0][0]
stage2_unit1_relu1 (Activation)	(None, 128, 128, 64)	0	stage2_unit1_bn1[0][0]
zero_padding2d_8 (ZeroPadding2D	(None, 130, 130, 64)	0	stage2_unit1_relu1[0][0]
stage2_unit1_conv1 (Conv2D)	(None, 64, 64, 128)	73728	zero_padding2d_8[0][0]
stage2_unit1_bn2 (BatchNormaliz	(None, 64, 64, 128)	512	stage2_unit1_conv1[0][0]
stage2_unit1_relu2 (Activation)	(None, 64, 64, 128)	0	stage2_unit1_bn2[0][0]
zero_padding2d_9 (ZeroPadding2D	(None, 66, 66, 128)	0	stage2_unit1_relu2[0][0]
stage2_unit1_conv2 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_9[0][0]
stage2_unit1_sc (Conv2D)	(None, 64, 64, 128)	8192	stage2_unit1_relu1[0][0]
add_3 (Add)	(None, 64, 64, 128)	0	stage2_unit1_conv2[0][0] stage2_unit1_sc[0][0]
stage2_unit2_bn1 (BatchNormaliz	(None, 64, 64, 128)	512	add_3[0][0]
stage2_unit2_relu1 (Activation)	(None, 64, 64, 128)	0	stage2_unit2_bn1[0][0]
zero_padding2d_10 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit2_relu1[0][0]
stage2_unit2_conv1 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_10[0][0]
stage2_unit2_bn2 (BatchNormaliz	(None, 64, 64, 128)	512	stage2_unit2_conv1[0][0]
stage2_unit2_relu2 (Activation)	(None, 64, 64, 128)	0	stage2_unit2_bn2[0][0]
zero_padding2d_11 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit2_relu2[0][0]
stage2_unit2_conv2 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_11[0][0]
add_4 (Add)	(None, 64, 64, 128)	0	stage2_unit2_conv2[0][0] add_3[0][0]
stage2_unit3_bn1 (BatchNormaliz	(None, 64, 64, 128)	512	add_4[0][0]
stage2_unit3_relu1 (Activation)	(None, 64, 64, 128)	0	stage2_unit3_bn1[0][0]

zero_padding2d_12 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit3_relu1[0][0]
stage2_unit3_conv1 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_12[0][0]
stage2_unit3_bn2 (BatchNormaliz	(None, 64, 64, 128)	512	stage2_unit3_conv1[0][0]
stage2_unit3_relu2 (Activation)	(None, 64, 64, 128)	0	stage2_unit3_bn2[0][0]
zero_padding2d_13 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit3_relu2[0][0]
stage2_unit3_conv2 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_13[0][0]
add_5 (Add)	(None, 64, 64, 128)	0	stage2_unit3_conv2[0][0] add_4[0][0]
stage2_unit4_bn1 (BatchNormaliz	(None, 64, 64, 128)	512	add_5[0][0]
stage2_unit4_relu1 (Activation)	(None, 64, 64, 128)	0	stage2_unit4_bn1[0][0]
zero_padding2d_14 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit4_relu1[0][0]
stage2_unit4_conv1 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_14[0][0]
stage2_unit4_bn2 (BatchNormaliz	(None, 64, 64, 128)	512	stage2_unit4_conv1[0][0]
stage2_unit4_relu2 (Activation)	(None, 64, 64, 128)	0	stage2_unit4_bn2[0][0]
zero_padding2d_15 (ZeroPadding2	(None, 66, 66, 128)	0	stage2_unit4_relu2[0][0]
stage2_unit4_conv2 (Conv2D)	(None, 64, 64, 128)	147456	zero_padding2d_15[0][0]
add_6 (Add)	(None, 64, 64, 128)	0	stage2_unit4_conv2[0][0] add_5[0][0]
stage3_unit1_bn1 (BatchNormaliz	(None, 64, 64, 128)	512	add_6[0][0]
stage3_unit1_relu1 (Activation)	(None, 64, 64, 128)	0	stage3_unit1_bn1[0][0]
zero_padding2d_16 (ZeroPadding2	(None, 66, 66, 128)	0	stage3_unit1_relu1[0][0]
stage3_unit1_conv1 (Conv2D)	(None, 32, 32, 256)	294912	zero_padding2d_16[0][0]
stage3_unit1_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit1_conv1[0][0]
stage3_unit1_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit1_bn2[0][0]
zero_padding2d_17 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit1_relu2[0][0]
stage3_unit1_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_17[0][0]
stage3_unit1_sc (Conv2D)	(None, 32, 32, 256)	32768	stage3_unit1_relu1[0][0]
add_7 (Add)	(None, 32, 32, 256)	0	stage3_unit1_conv2[0][0] stage3_unit1_sc[0][0]
stage3_unit2_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_7[0][0]
stage3_unit2_relu1 (Activation)	(None, 32, 32, 256)	0	stage3_unit2_bn1[0][0]
zero_padding2d_18 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit2_relu1[0][0]
stage3_unit2_conv1 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_18[0][0]
stage3_unit2_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit2_conv1[0][0]
stage3_unit2_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit2_bn2[0][0]
zero_padding2d_19 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit2_relu2[0][0]
stage3_unit2_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_19[0][0]
add_8 (Add)	(None, 32, 32, 256)	0	stage3_unit2_conv2[0][0] add_7[0][0]
stage3_unit3_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_8[0][0]
stage3_unit3_relu1 (Activation)	(None, 32, 32, 256)	0	stage3_unit3_bn1[0][0]
zero_padding2d_20 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit3_relu1[0][0]
stage3_unit3_conv1 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_20[0][0]
stage3_unit3_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit3_conv1[0][0]
stage3_unit3_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit3_bn2[0][0]
zero_padding2d_21 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit3_relu2[0][0]
stage3_unit3_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_21[0][0]
add_9 (Add)	(None, 32, 32, 256)	0	stage3_unit3_conv2[0][0] add_8[0][0]
stage3_unit4_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_9[0][0]
stage3_unit4_relu1 (Activation)	(None, 32, 32, 256)	0	stage3_unit4_bn1[0][0]
zero_padding2d_22 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit4_relu1[0][0]

stage3_unit4_conv1 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_22[0][0]
stage3_unit4_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit4_conv1[0][0]
stage3_unit4_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit4_bn2[0][0]
zero_padding2d_23 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit4_relu2[0][0]
stage3_unit4_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_23[0][0]
add_10 (Add)	(None, 32, 32, 256)	0	stage3_unit4_conv2[0][0] add_9[0][0]
stage3_unit5_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_10[0][0]
stage3_unit5_relu1 (Activation)	(None, 32, 32, 256)	0	stage3_unit5_bn1[0][0]
zero_padding2d_24 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit5_relu1[0][0]
stage3_unit5_conv1 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_24[0][0]
stage3_unit5_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit5_conv1[0][0]
stage3_unit5_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit5_bn2[0][0]
zero_padding2d_25 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit5_relu2[0][0]
stage3_unit5_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_25[0][0]
add_11 (Add)	(None, 32, 32, 256)	0	stage3_unit5_conv2[0][0] add_10[0][0]
stage3_unit6_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_11[0][0]
stage3_unit6_relu1 (Activation)	(None, 32, 32, 256)	0	stage3_unit6_bn1[0][0]
zero_padding2d_26 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit6_relu1[0][0]
stage3_unit6_conv1 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_26[0][0]
stage3_unit6_bn2 (BatchNormaliz	(None, 32, 32, 256)	1024	stage3_unit6_conv1[0][0]
stage3_unit6_relu2 (Activation)	(None, 32, 32, 256)	0	stage3_unit6_bn2[0][0]
zero_padding2d_27 (ZeroPadding2	(None, 34, 34, 256)	0	stage3_unit6_relu2[0][0]
stage3_unit6_conv2 (Conv2D)	(None, 32, 32, 256)	589824	zero_padding2d_27[0][0]
add_12 (Add)	(None, 32, 32, 256)	0	stage3_unit6_conv2[0][0] add_11[0][0]
stage4_unit1_bn1 (BatchNormaliz	(None, 32, 32, 256)	1024	add_12[0][0]
stage4_unit1_relu1 (Activation)	(None, 32, 32, 256)	0	stage4_unit1_bn1[0][0]
zero_padding2d_28 (ZeroPadding2	(None, 34, 34, 256)	0	stage4_unit1_relu1[0][0]
stage4_unit1_conv1 (Conv2D)	(None, 16, 16, 512)	1179648	zero_padding2d_28[0][0]
stage4_unit1_bn2 (BatchNormaliz	(None, 16, 16, 512)	2048	stage4_unit1_conv1[0][0]
stage4_unit1_relu2 (Activation)	(None, 16, 16, 512)	0	stage4_unit1_bn2[0][0]
zero_padding2d_29 (ZeroPadding2	(None, 18, 18, 512)	0	stage4_unit1_relu2[0][0]
stage4_unit1_conv2 (Conv2D)	(None, 16, 16, 512)	2359296	zero_padding2d_29[0][0]
stage4_unit1_sc (Conv2D)	(None, 16, 16, 512)	131072	stage4_unit1_relu1[0][0]
add_13 (Add)	(None, 16, 16, 512)	0	stage4_unit1_conv2[0][0] stage4_unit1_sc[0][0]
stage4_unit2_bn1 (BatchNormaliz	(None, 16, 16, 512)	2048	add_13[0][0]
stage4_unit2_relu1 (Activation)	(None, 16, 16, 512)	0	stage4_unit2_bn1[0][0]
zero_padding2d_30 (ZeroPadding2	(None, 18, 18, 512)	0	stage4_unit2_relu1[0][0]
stage4_unit2_conv1 (Conv2D)	(None, 16, 16, 512)	2359296	zero_padding2d_30[0][0]
stage4_unit2_bn2 (BatchNormaliz	(None, 16, 16, 512)	2048	stage4_unit2_conv1[0][0]
stage4_unit2_relu2 (Activation)	(None, 16, 16, 512)	0	stage4_unit2_bn2[0][0]
zero_padding2d_31 (ZeroPadding2	(None, 18, 18, 512)	0	stage4_unit2_relu2[0][0]
stage4_unit2_conv2 (Conv2D)	(None, 16, 16, 512)	2359296	zero_padding2d_31[0][0]
add_14 (Add)	(None, 16, 16, 512)	0	stage4_unit2_conv2[0][0] add_13[0][0]
stage4_unit3_bn1 (BatchNormaliz	(None, 16, 16, 512)	2048	add_14[0][0]
stage4_unit3_relu1 (Activation)	(None, 16, 16, 512)	0	stage4_unit3_bn1[0][0]
zero_padding2d_32 (ZeroPadding2	(None, 18, 18, 512)	0	stage4_unit3_relu1[0][0]

stage4_unit3_conv1 (Conv2D)	(None, 16, 16, 512)	2359296	zero_padding2d_32[0][0]
stage4_unit3_bn2 (BatchNormaliz	(None, 16, 16, 512)	2048	stage4_unit3_conv1[0][0]
stage4_unit3_relu2 (Activation)	(None, 16, 16, 512)	0	stage4_unit3_bn2[0][0]
zero_padding2d_33 (ZeroPadding2	(None, 18, 18, 512)	0	stage4_unit3_relu2[0][0]
stage4_unit3_conv2 (Conv2D)	(None, 16, 16, 512)	2359296	zero_padding2d_33[0][0]
add_15 (Add)	(None, 16, 16, 512)	0	stage4_unit3_conv2[0][0] add_14[0][0]
bn1 (BatchNormalization)	(None, 16, 16, 512)	2048	add_15[0][0]
relu1 (Activation)	(None, 16, 16, 512)	0	bn1[0][0]
decoder_stage0_upsampling (UpSa	(None, 32, 32, 512)	0	relu1[0][0]
decoder_stage0_concat (Concaten	(None, 32, 32, 768)	0	decoder_stage0_upsampling[0][0] stage4_unit1_relu1[0][0]
decoder_stage0a_conv (Conv2D)	(None, 32, 32, 256)	1769472	decoder_stage0_concat[0][0]
decoder_stage0a_bn (BatchNormal	(None, 32, 32, 256)	1024	decoder_stage0a_conv[0][0]
decoder_stage0a_relu (Activatio	(None, 32, 32, 256)	0	decoder_stage0a_bn[0][0]
decoder_stage0b_conv (Conv2D)	(None, 32, 32, 256)	589824	decoder_stage0a_relu[0][0]
decoder_stage0b_bn (BatchNormal	(None, 32, 32, 256)	1024	decoder_stage0b_conv[0][0]
decoder_stage0b_relu (Activatio	(None, 32, 32, 256)	0	decoder_stage0b_bn[0][0]
decoder_stage1_upsampling (UpSa	(None, 64, 64, 256)	0	decoder_stage0b_relu[0][0]
decoder_stage1_concat (Concaten	(None, 64, 64, 384)	0	decoder_stage1_upsampling[0][0] stage3_unit1_relu1[0][0]
decoder_stage1a_conv (Conv2D)	(None, 64, 64, 128)	442368	decoder_stage1_concat[0][0]
decoder_stage1a_bn (BatchNormal	(None, 64, 64, 128)	512	decoder_stage1a_conv[0][0]
decoder_stage1a_relu (Activatio	(None, 64, 64, 128)	0	decoder_stage1a_bn[0][0]
decoder_stage1b_conv (Conv2D)	(None, 64, 64, 128)	147456	decoder_stage1a_relu[0][0]
decoder_stage1b_bn (BatchNormal	(None, 64, 64, 128)	512	decoder_stage1b_conv[0][0]
decoder_stage1b_relu (Activatio	(None, 64, 64, 128)	0	decoder_stage1b_bn[0][0]
decoder_stage2_upsampling (UpSa	(None, 128, 128, 128	0	decoder_stage1b_relu[0][0]
decoder_stage2_concat (Concaten	(None, 128, 128, 192	0	decoder_stage2_upsampling[0][0] stage2_unit1_relu1[0][0]
decoder_stage2a_conv (Conv2D)	(None, 128, 128, 64)	110592	decoder_stage2_concat[0][0]
decoder_stage2a_bn (BatchNormal	(None, 128, 128, 64)	256	decoder_stage2a_conv[0][0]
decoder_stage2a_relu (Activatio	(None, 128, 128, 64)	0	decoder_stage2a_bn[0][0]
decoder_stage2b_conv (Conv2D)	(None, 128, 128, 64)	36864	decoder_stage2a_relu[0][0]
decoder_stage2b_bn (BatchNormal	(None, 128, 128, 64)	256	decoder_stage2b_conv[0][0]
decoder_stage2b_relu (Activatio	(None, 128, 128, 64)	0	decoder_stage2b_bn[0][0]
decoder_stage3_upsampling (UpSa	(None, 256, 256, 64)	0	decoder_stage2b_relu[0][0]
decoder_stage3_concat (Concaten	(None, 256, 256, 128	0	decoder_stage3_upsampling[0][0] relu0[0][0]
decoder_stage3a_conv (Conv2D)	(None, 256, 256, 32)	36864	decoder_stage3_concat[0][0]
decoder_stage3a_bn (BatchNormal	(None, 256, 256, 32)	128	decoder_stage3a_conv[0][0]
decoder_stage3a_relu (Activatio	(None, 256, 256, 32)	0	decoder_stage3a_bn[0][0]
decoder_stage3b_conv (Conv2D)	(None, 256, 256, 32)	9216	decoder_stage3a_relu[0][0]
decoder_stage3b_bn (BatchNormal	(None, 256, 256, 32)	128	decoder_stage3b_conv[0][0]
decoder_stage3b_relu (Activatio	(None, 256, 256, 32)	0	decoder_stage3b_bn[0][0]
decoder_stage4_upsampling (UpSa	(None, 512, 512, 32)	0	decoder_stage3b_relu[0][0]
decoder_stage4a_conv (Conv2D)	(None, 512, 512, 16)	4608	decoder_stage4_upsampling[0][0]
decoder_stage4a_bn (BatchNormal	(None, 512, 512, 16)	64	decoder_stage4a_conv[0][0]
decoder_stage4a_relu (Activatio	(None, 512, 512, 16)	0	decoder_stage4a_bn[0][0]
decoder_stage4b_conv (Conv2D)	(None, 512, 512, 16)	2304	decoder_stage4a_relu[0][0]
decoder_stage4b_bn (BatchNormal	(None, 512, 512, 16)	64	decoder_stage4b_conv[0][0]
decoder_stage4b_relu (Activatio	(None, 512, 512, 16)	0	decoder_stage4b_bn[0][0]

```
final_conv (Conv2D)             (None, 512, 512, 21) 3045        decoder_stage4b_relu[0][0]
softmax (Activation)            (None, 512, 512, 21) 0           final_conv[0][0]
Total params: 24,459,054
Trainable params: 3,169,960
Non-trainable params: 21,289,094
```

In [ ]:
```
import imgaug.augmenters as iaa
# For the assignment choose any 4 augumentation techniques
# check the imgaug documentations for more augmentations
aug2 = iaa.Fliplr(1)
aug3 = iaa.Flipud(1)
aug4 = iaa.Emboss(alpha=(1), strength=1)
aug5 = iaa.DirectedEdgeDetect(alpha=(0.8), direction=(1.0))
aug6 = iaa.Sharpen(alpha=(1.0), lightness=(1.5))
```

In [ ]:
```
class Dataset:
    def __init__(self, file_names, classes , types):

        self.ids = file_names
        self.class_ids = classes
        self.types = types
        self.images_fps    = [image_id[0] for image_id in self.ids]
        self.masks_fps     = [image_id[1] for image_id in self.ids]



    def __getitem__(self, i):

        # read data
        image = cv2.imread(self.images_fps[i], cv2.IMREAD_UNCHANGED)
        image = np.float32(cv2.resize(image, (512, 512)))

        mask  = cv2.imread(self.masks_fps[i], cv2.IMREAD_UNCHANGED)
        image_mask = mask


        image_masks = [(image_mask == i) for i  in self.class_ids]
        image_mask = np.stack(image_masks, axis=-1).astype('float')
        image_mask = np.float32(cv2.resize(image_mask, (512, 512)))


        if self.types == 'train':
          a = np.random.uniform()
          if a<0.2:
            image = aug2.augment_image(image)
            image_mask = aug2.augment_image(image_mask)
          elif a<0.4:
            image = aug3.augment_image(image)
            image_mask = aug3.augment_image(image_mask)
          elif a<0.6:
            image = aug4.augment_image(image)
            image_mask = aug4.augment_image(image_mask)
          elif a<0.8:
            image = aug5.augment_image(image)
            image_mask = image_mask
          else:
            image = aug6.augment_image(image)
            image_mask = aug6.augment_image(image_mask)


          return image, image_mask

        else:

          return image, image_mask


    def __len__(self):
        return len(self.ids)


class Dataloder(tf.keras.utils.Sequence):
    def __init__(self, dataset, batch_size=1, shuffle=False):
        self.dataset = dataset
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.indexes = np.arange(len(dataset))

    def __getitem__(self, i):

        # collect batch data
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
        data = []
        for j in range(start, stop):
            data.append(self.dataset[j])
```

```
        batch = [np.stack(samples, axis=0) for samples in zip(*data)]

        return tuple(batch)

    def __len__(self):
        return len(self.indexes) // self.batch_size

    def on_epoch_end(self):
        if self.shuffle:
            self.indexes = np.random.permutation(self.indexes)
```

In [ ]:
```
# https://github.com/qubvel/segmentation_models
#import segmentation_models as sm
from segmentation_models.metrics import iou_score



rms = tf.keras.optimizers.RMSprop(learning_rate=0.001)

focal_loss = sm.losses.cce_dice_loss

model.compile(rms, focal_loss, metrics=[iou_score])
```

In [ ]:
```
#CLASSES = CLASSES
train_dataset = Dataset(X_train, classes=class_label , types ='train')
test_dataset  = Dataset(X_test, classes=class_label, types = 'test')

train_dataloader = Dataloder(train_dataset, batch_size=5, shuffle=True)
test_dataloader = Dataloder(test_dataset, batch_size=5, shuffle=True)

print(train_dataloader[0][0].shape)
assert train_dataloader[0][0].shape == (5, 512, 512, 3)
assert train_dataloader[0][1].shape == (5, 512, 512, 21)
```

```
(5, 512, 512, 3)
```

In [ ]:
```
(train_dataloader[0][0].shape) , (train_dataloader[0][1].shape)
```

Out[ ]:
```
((5, 512, 512, 3), (5, 512, 512, 21))
```

In [ ]:
```
(test_dataloader[0][0].shape) , (test_dataloader[0][1].shape)
```

Out[ ]:
```
((5, 512, 512, 3), (5, 512, 512, 21))
```

In [ ]:
```
callbacks_ = [
    tf.keras.callbacks.ModelCheckpoint('./best_model.h5', save_weights_only=True, save_best_only=True, \
                                       mode='min', monitor='val_iou_score')]
```

In [ ]:
```
history = model.fit(train_dataloader, steps_per_epoch=len(train_dataloader)//5, epochs=70,\
                              validation_data=test_dataloader,validation_steps =len(test_dataloader)//5,callbacks=callbacks_ )
```

```
Epoch 1/70
136/136 [==============================] - 1277s 9s/step - loss: 0.9502 - iou_score: 0.0928 - val_loss: 1.0563 - val_iou_score: 0.0939
Epoch 2/70
136/136 [==============================] - 713s 5s/step - loss: 0.8116 - iou_score: 0.1841 - val_loss: 0.8965 - val_iou_score: 0.1386
Epoch 3/70
136/136 [==============================] - 625s 5s/step - loss: 0.7488 - iou_score: 0.2300 - val_loss: 0.8317 - val_iou_score: 0.1837
Epoch 4/70
136/136 [==============================] - 608s 4s/step - loss: 0.7204 - iou_score: 0.2569 - val_loss: 0.7799 - val_iou_score: 0.2259
Epoch 5/70
136/136 [==============================] - 568s 4s/step - loss: 0.6035 - iou_score: 0.3684 - val_loss: 0.5956 - val_iou_score: 0.3743
Epoch 6/70
136/136 [==============================] - 568s 4s/step - loss: 0.5926 - iou_score: 0.3792 - val_loss: 0.5894 - val_iou_score: 0.3812
Epoch 7/70
136/136 [==============================] - 517s 4s/step - loss: 0.5836 - iou_score: 0.3844 - val_loss: 0.5485 - val_iou_score: 0.4148
Epoch 8/70
136/136 [==============================] - 496s 4s/step - loss: 0.5613 - iou_score: 0.4018 - val_loss: 0.5574 - val_iou_score: 0.4091
Epoch 9/70
136/136 [==============================] - 483s 4s/step - loss: 0.5534 - iou_score: 0.4059 - val_loss: 0.5678 - val_iou_score: 0.4012
Epoch 10/70
136/136 [==============================] - 465s 3s/step - loss: 0.5456 - iou_score: 0.4156 - val_loss: 0.6066 - val_iou_score: 0.3757
Epoch 11/70
136/136 [==============================] - 463s 3s/step - loss: 0.5351 - iou_score: 0.4220 - val_loss: 0.6443 - val_iou_score: 0.3549
Epoch 12/70
136/136 [==============================] - 457s 3s/step - loss: 0.5423 - iou_score: 0.4185 - val_loss: 0.6622 - val_iou_score: 0.3486
Epoch 13/70
136/136 [==============================] - 451s 3s/step - loss: 0.5224 - iou_score: 0.4343 - val_loss: 0.6430 - val_iou_score: 0.3587
Epoch 14/70
136/136 [==============================] - 423s 3s/step - loss: 0.5178 - iou_score: 0.4357 - val_loss: 0.5280 - val_iou_score: 0.4332
Epoch 15/70
136/136 [==============================] - 445s 3s/step - loss: 0.5086 - iou_score: 0.4483 - val_loss: 0.5199 - val_iou_score: 0.4410
Epoch 16/70
136/136 [==============================] - 445s 3s/step - loss: 0.5025 - iou_score: 0.4542 - val_loss: 0.5134 - val_iou_score: 0.4429
Epoch 17/70
136/136 [==============================] - 425s 3s/step - loss: 0.5059 - iou_score: 0.4481 - val_loss: 0.5189 - val_iou_score: 0.4407
Epoch 18/70
136/136 [==============================] - 437s 3s/step - loss: 0.4964 - iou_score: 0.4582 - val_loss: 0.4921 - val_iou_score: 0.4611
Epoch 19/70
136/136 [==============================] - 439s 3s/step - loss: 0.4878 - iou_score: 0.4668 - val_loss: 0.5042 - val_iou_score: 0.4502
Epoch 20/70
136/136 [==============================] - 439s 3s/step - loss: 0.4894 - iou_score: 0.4654 - val_loss: 0.5027 - val_iou_score: 0.4566
```

```
Epoch 21/70
136/136 [==============================] - 420s 3s/step - loss: 0.4903 - iou_score: 0.4625 - val_loss: 0.4796 - val_iou_score: 0.4751
Epoch 22/70
136/136 [==============================] - 427s 3s/step - loss: 0.4886 - iou_score: 0.4629 - val_loss: 0.4836 - val_iou_score: 0.4687
Epoch 23/70
136/136 [==============================] - 420s 3s/step - loss: 0.4800 - iou_score: 0.4773 - val_loss: 0.5087 - val_iou_score: 0.4525
Epoch 24/70
136/136 [==============================] - 419s 3s/step - loss: 0.4780 - iou_score: 0.4737 - val_loss: 0.4844 - val_iou_score: 0.4690
Epoch 25/70
136/136 [==============================] - 414s 3s/step - loss: 0.4672 - iou_score: 0.4884 - val_loss: 0.4714 - val_iou_score: 0.4795
Epoch 26/70
136/136 [==============================] - 417s 3s/step - loss: 0.4820 - iou_score: 0.4684 - val_loss: 0.4769 - val_iou_score: 0.4738
Epoch 27/70
136/136 [==============================] - 416s 3s/step - loss: 0.4751 - iou_score: 0.4795 - val_loss: 0.4794 - val_iou_score: 0.4746
Epoch 28/70
136/136 [==============================] - 424s 3s/step - loss: 0.4742 - iou_score: 0.4750 - val_loss: 0.4737 - val_iou_score: 0.4790
Epoch 29/70
136/136 [==============================] - 418s 3s/step - loss: 0.4613 - iou_score: 0.4944 - val_loss: 0.4916 - val_iou_score: 0.4558
Epoch 30/70
136/136 [==============================] - 408s 3s/step - loss: 0.4720 - iou_score: 0.4816 - val_loss: 0.4839 - val_iou_score: 0.4709
Epoch 31/70
136/136 [==============================] - 420s 3s/step - loss: 0.4671 - iou_score: 0.4826 - val_loss: 0.4698 - val_iou_score: 0.4859
Epoch 32/70
136/136 [==============================] - 418s 3s/step - loss: 0.4684 - iou_score: 0.4860 - val_loss: 0.4802 - val_iou_score: 0.4761
Epoch 33/70
136/136 [==============================] - 416s 3s/step - loss: 0.4605 - iou_score: 0.4905 - val_loss: 0.4510 - val_iou_score: 0.5000
Epoch 34/70
136/136 [==============================] - 419s 3s/step - loss: 0.4560 - iou_score: 0.4965 - val_loss: 0.4570 - val_iou_score: 0.4952
Epoch 35/70
136/136 [==============================] - 419s 3s/step - loss: 0.4468 - iou_score: 0.5061 - val_loss: 0.4582 - val_iou_score: 0.4951
Epoch 36/70
136/136 [==============================] - 410s 3s/step - loss: 0.4393 - iou_score: 0.5103 - val_loss: 0.4619 - val_iou_score: 0.4904
Epoch 37/70
136/136 [==============================] - 419s 3s/step - loss: 0.4603 - iou_score: 0.5048 - val_loss: 0.4519 - val_iou_score: 0.5002
Epoch 38/70
136/136 [==============================] - 417s 3s/step - loss: 0.4509 - iou_score: 0.4996 - val_loss: 0.4512 - val_iou_score: 0.5015
Epoch 39/70
136/136 [==============================] - 417s 3s/step - loss: 0.4315 - iou_score: 0.5214 - val_loss: 0.4584 - val_iou_score: 0.4936
Epoch 40/70
136/136 [==============================] - 412s 3s/step - loss: 0.4458 - iou_score: 0.5057 - val_loss: 0.4496 - val_iou_score: 0.5037
Epoch 41/70
136/136 [==============================] - 416s 3s/step - loss: 0.4462 - iou_score: 0.5047 - val_loss: 0.4532 - val_iou_score: 0.4958
Epoch 42/70
136/136 [==============================] - 419s 3s/step - loss: 0.4463 - iou_score: 0.5069 - val_loss: 0.4479 - val_iou_score: 0.5054
Epoch 43/70
136/136 [==============================] - 419s 3s/step - loss: 0.4553 - iou_score: 0.4939 - val_loss: 0.4560 - val_iou_score: 0.5009
Epoch 44/70
136/136 [==============================] - 425s 3s/step - loss: 0.4531 - iou_score: 0.4994 - val_loss: 0.4498 - val_iou_score: 0.5012
Epoch 45/70
136/136 [==============================] - 433s 3s/step - loss: 0.4502 - iou_score: 0.5014 - val_loss: 0.4593 - val_iou_score: 0.4921
Epoch 46/70
136/136 [==============================] - 426s 3s/step - loss: 0.4427 - iou_score: 0.5099 - val_loss: 0.4439 - val_iou_score: 0.5061
Epoch 47/70
136/136 [==============================] - 429s 3s/step - loss: 0.4339 - iou_score: 0.5193 - val_loss: 0.4447 - val_iou_score: 0.5090
Epoch 48/70
136/136 [==============================] - 432s 3s/step - loss: 0.4508 - iou_score: 0.4986 - val_loss: 0.4447 - val_iou_score: 0.5061
Epoch 49/70
136/136 [==============================] - 426s 3s/step - loss: 0.4240 - iou_score: 0.5222 - val_loss: 0.4431 - val_iou_score: 0.5074
Epoch 50/70
136/136 [==============================] - 432s 3s/step - loss: 0.4410 - iou_score: 0.5108 - val_loss: 0.4445 - val_iou_score: 0.5069
Epoch 51/70
136/136 [==============================] - 427s 3s/step - loss: 0.4371 - iou_score: 0.5133 - val_loss: 0.4349 - val_iou_score: 0.5174
Epoch 52/70
136/136 [==============================] - 420s 3s/step - loss: 0.4353 - iou_score: 0.5122 - val_loss: 0.4496 - val_iou_score: 0.5007
Epoch 53/70
136/136 [==============================] - 432s 3s/step - loss: 0.4399 - iou_score: 0.5109 - val_loss: 0.4388 - val_iou_score: 0.5185
Epoch 54/70
136/136 [==============================] - 427s 3s/step - loss: 0.4339 - iou_score: 0.5160 - val_loss: 0.4333 - val_iou_score: 0.5158
Epoch 55/70
136/136 [==============================] - 431s 3s/step - loss: 0.4370 - iou_score: 0.5139 - val_loss: 0.4344 - val_iou_score: 0.5174
Epoch 56/70
136/136 [==============================] - 432s 3s/step - loss: 0.4265 - iou_score: 0.5219 - val_loss: 0.4373 - val_iou_score: 0.5117
Epoch 57/70
136/136 [==============================] - 433s 3s/step - loss: 0.4401 - iou_score: 0.5080 - val_loss: 0.4362 - val_iou_score: 0.5135
Epoch 58/70
136/136 [==============================] - 435s 3s/step - loss: 0.4260 - iou_score: 0.5232 - val_loss: 0.4412 - val_iou_score: 0.5110
Epoch 59/70
136/136 [==============================] - 430s 3s/step - loss: 0.4366 - iou_score: 0.5134 - val_loss: 0.4290 - val_iou_score: 0.5212
Epoch 60/70
136/136 [==============================] - 435s 3s/step - loss: 0.4345 - iou_score: 0.5183 - val_loss: 0.4359 - val_iou_score: 0.5190
Epoch 61/70
136/136 [==============================] - 426s 3s/step - loss: 0.4271 - iou_score: 0.5251 - val_loss: 0.4325 - val_iou_score: 0.5196
Epoch 62/70
136/136 [==============================] - 428s 3s/step - loss: 0.4416 - iou_score: 0.5091 - val_loss: 0.4291 - val_iou_score: 0.5219
Epoch 63/70
136/136 [==============================] - 430s 3s/step - loss: 0.4379 - iou_score: 0.5122 - val_loss: 0.4365 - val_iou_score: 0.5148
Epoch 64/70
136/136 [==============================] - 431s 3s/step - loss: 0.4196 - iou_score: 0.5319 - val_loss: 0.4223 - val_iou_score: 0.5303
Epoch 65/70
136/136 [==============================] - 431s 3s/step - loss: 0.4253 - iou_score: 0.5269 - val_loss: 0.4292 - val_iou_score: 0.5209
Epoch 66/70
136/136 [==============================] - 413s 3s/step - loss: 0.4225 - iou_score: 0.5273 - val_loss: 0.4383 - val_iou_score: 0.5152
Epoch 67/70
136/136 [==============================] - 411s 3s/step - loss: 0.4139 - iou_score: 0.5340 - val_loss: 0.4338 - val_iou_score: 0.5145
Epoch 68/70
136/136 [==============================] - 413s 3s/step - loss: 0.4229 - iou_score: 0.5305 - val_loss: 0.4331 - val_iou_score: 0.5187
Epoch 69/70
136/136 [==============================] - 411s 3s/step - loss: 0.4225 - iou_score: 0.5293 - val_loss: 0.4296 - val_iou_score: 0.5169
```

```
Epoch 70/70
136/136 [==============================] - 413s 3s/step - loss: 0.4263 - iou_score: 0.5228 - val_loss: 0.4215 - val_iou_score: 0.5318
```

Task 2.1: Dice loss

* Explain the Dice loss ?
ans :- Dice loss is the measure of overlap between the predict sample and the srue sample. suppose we have true sample pixel size of 512 * 512 and predict sample size of 512 * 512 , then dice loss function compare the each pixel from both of them. and calculate the average value which range between 0 to 1. 0 define for there is no similarity between the true and predicted sample and 1 define both true and predicted sample are completely overlapping.

* 1. Write the formualtion ?
ans :- Dice loss = 2*sum of (product of each pixel from true and predicted sample) from 1 to n divided by sum of square of true and predicted pixel from 1 to n where n is the number of pixel.

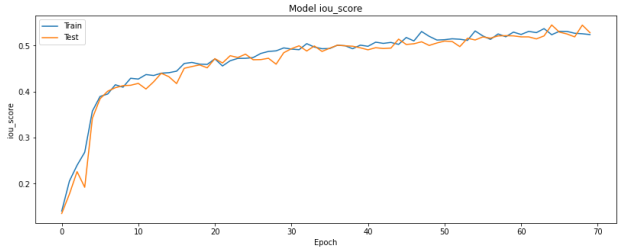
mathematicaly = 2 * ∑(tp * pp) from 1 to n/(∑square(tp) + ∑square(pp)) from 1 to n.

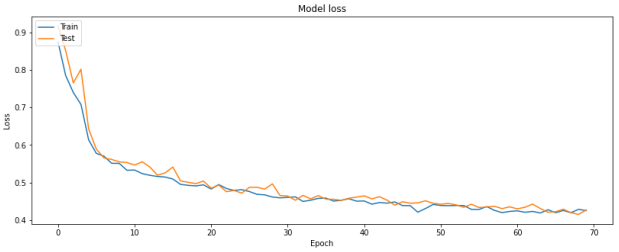
* 2. Range of the loss function ?
ans :- it has the range between 0 to 1.

* 3. Interpretation of loss function ?
ans:- 0 define for there is no similarity between the true and predicted sample and 1 define both true and predicted sample are completely overlapping.

* 4. Write your understanding of the loss function, how does it helps in segmentation ?
ans :-since we are predicting the pixel of image as an output. it is necessary that the each pixel from both true and predicted sample should match. As from the dice formula, denomirator defines the sum of square of each pixel from both true and predicted pixel and numerator define the overlap between both pixel. if particular pixel from each sample is similar then only it make sence othrwise it is zero. So if our model is giving 100% result numerator and denomirator cancel each other. In practicle world this metric is very essential as getting highest accuracy is very dificult to get.

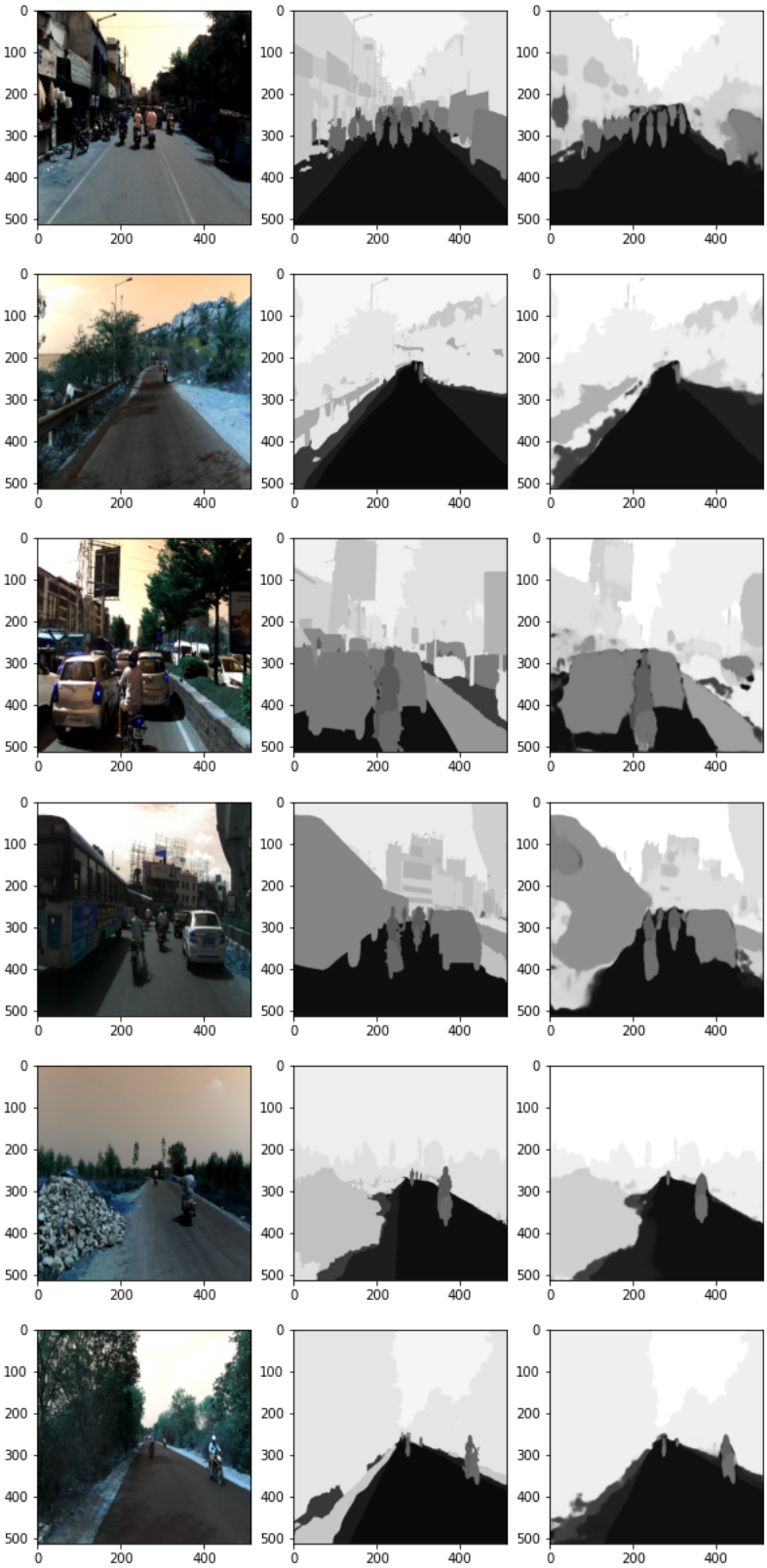
In [ ]:
```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['iou_score'])
plt.plot(history.history['val_iou_score'])
plt.title('Model iou_score')
plt.ylabel('iou_score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

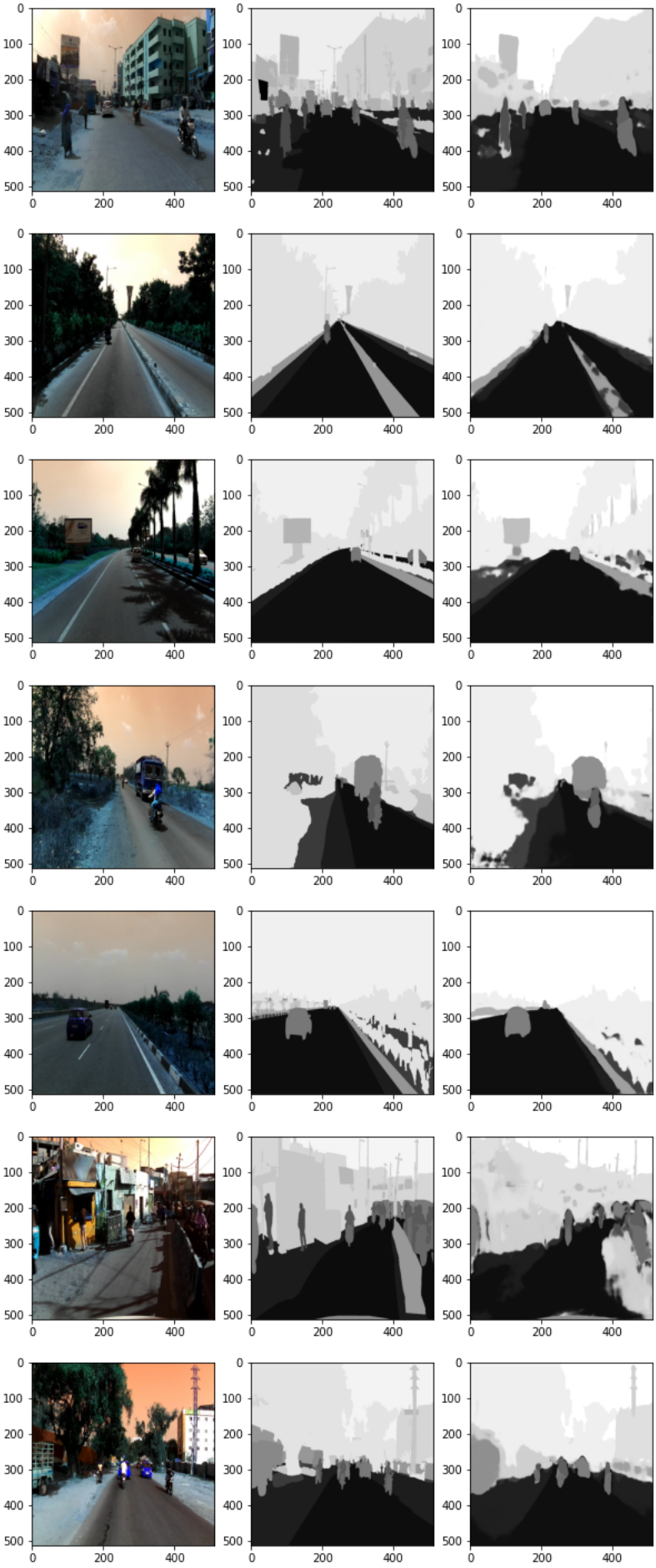


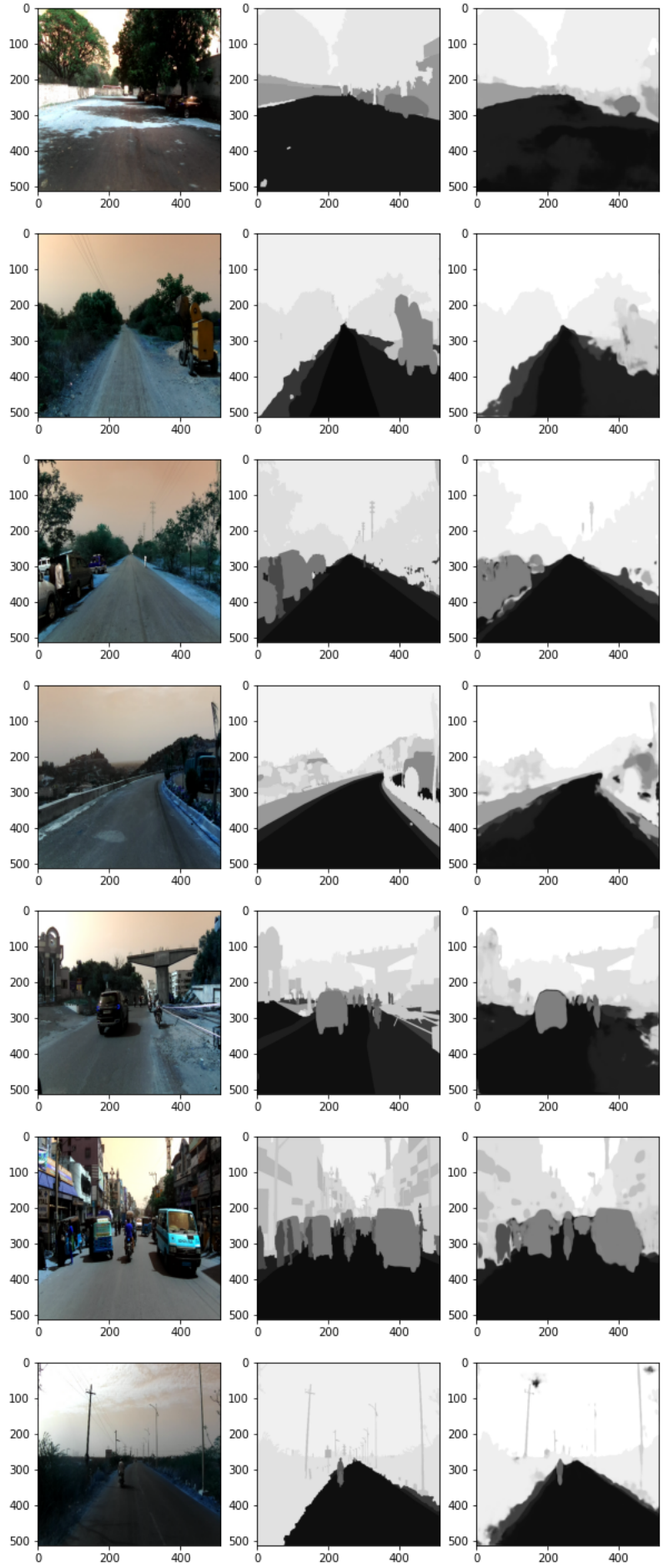


In [150…
```
a = X_test
for i in range(20):
    #original image
    image = cv2.imread(a[i][0],cv2.IMREAD_UNCHANGED)
    image = cv2.resize(image, (512, 512))

    #predicted segmentation map
    predicted  = model.predict(image[np.newaxis,:,:,:])
    predict = np.array([predicted[0,:,:,i]*j for i , j in enumerate(class_label)])
    predict = sum(predict)

    #original segmentation map
    image_mask = cv2.imread(a[i][1], cv2.IMREAD_UNCHANGED)
    image_mask = cv2.resize(image_mask, (512,512))


    plt.figure(figsize=(10,6))
    plt.subplot(131)
    plt.imshow(image)
    plt.subplot(132)
```

```
    plt.imshow(image_mask, cmap='gray')
    plt.subplot(133)
    plt.imshow(predict, cmap='gray')
    plt.show()
```







Task 2.2: Training Unet

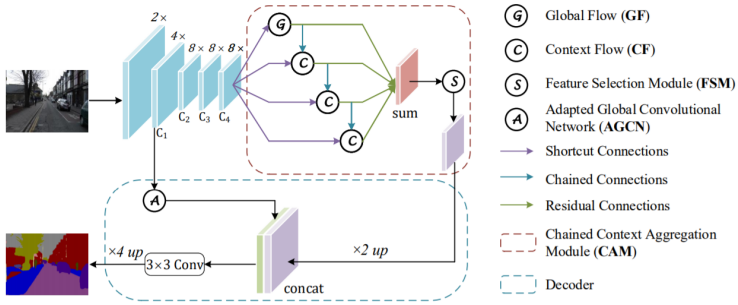
Task 3: Training CANet

In [158…

```
import tensorflow as tf
# tf.compat.v1.enable_eager_execution()
from tensorflow import keras
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import UpSampling2D
from tensorflow.keras.layers import MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Multiply
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, Max
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import glorot_uniform
K.set_image_data_format('channels_last')
K.set_learning_phase(1)
```

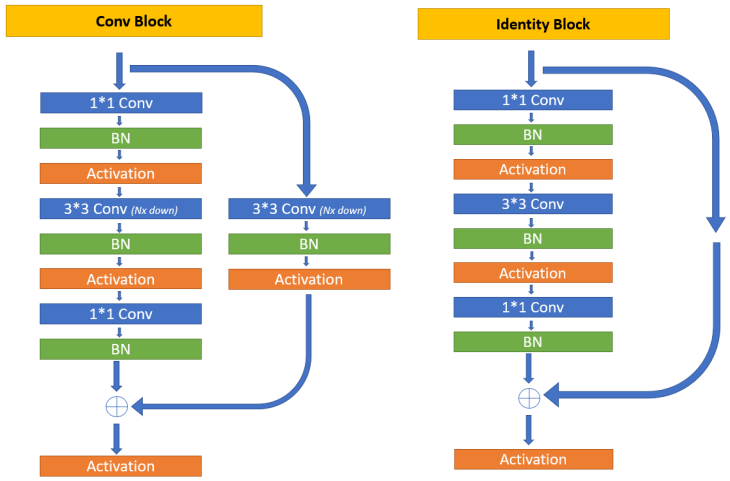
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:434: UserWarning: `tf.keras.backend.set_learning_phase` is depr
ecated and will be removed after 2020-10-11. To update it, simply pass a True/False value to the `training` argument of the `__call__` me
thod of your layer or model.
  warnings.warn('`tf.keras.backend.set_learning_phase` is deprecated and '
```

- as a part of this assignment we will be implementing the architecture based on this paper https://arxiv.org/pdf/2002.12041.pdf
- We will be using the custom layers concept that we used in seq-seq assignment
- You can devide the whole architecture can be devided into two parts
    1. Encoder



    2. Decoder

- Encoder:
    - The first step of the encoder is to create the channel maps $[C_1, C_2, C_3, C_4]$
    - C_1 width and heigths are 4x times less than the original image
    - C_2 width and heigths are 8x times less than the original image
    - C_3 width and heigths are 8x times less than the original image
    - C_4 width and heigths are 8x times less than the original image
    - *you can reduce the dimensions by using stride parameter*.
    - $[C_1, C_2, C_3, C_4]$ are formed by applying a "conv block" followed by k number of "identity block". i.e the C_k feature map will single "conv block" followed by k number of "identity blocks".



    - **The conv block and identity block of C_1**: the number filters in the covolutional layers will be $[4, 4, 8]$ and the number of filters in the parallel conv layer will also be 8.

- **The conv block and identity block of** C_2: the number filters in the covolutional layers will be $[8, 8, 16]$ and the number of filters in the parallel conv layer will also be 16.
- **The conv block and identity block of** C_3: the number filters in the covolutional layers will be $[16, 16, 32]$ and the number of filters in the parallel conv layer will also be 32.
- **The conv block and identity block of** C_4: the number filters in the covolutional layers will be $[32, 32, 64]$ and the number of filters in the parallel conv layer will also be 64.
- Here $\oplus$ represents the elementwise sum

  NOTE: these filters are of your choice, you can explore more options also

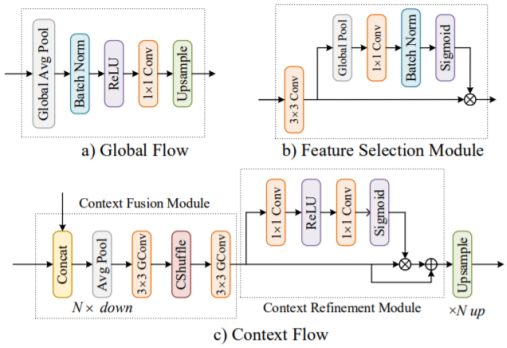
- Example: if your image is of size $(512, 512, 3)$
  - the output after C_1 will be $128 * 128 * 8$
  - the output after C_2 will be $64 * 64 * 16$
  - the output after C_3 will be $64 * 64 * 32$
  - the output after C_4 will be $64 * 64 * 64$

In [184…
```
class convolutional_block(tf.keras.layers.Layer):
    def __init__(self, kernel=3,  filters=[4,4,8], stride=1, name="conv_block"):
        super().__init__(name=name)
        self.F1, self.F2, self.F3 = filters
        self.kernel = kernel
        self.stride = stride

        self.Conv2d1 = Conv2D(self.F1, kernel_size =(1, 1), strides = (self.stride,self.stride),
                 kernel_initializer = glorot_uniform(seed=0))

        self.batch1 = BatchNormalization(axis = 3)
        self.batch2 = BatchNormalization(axis = 3)
        self.batch3 = BatchNormalization(axis = 3)
        self.batch4 = BatchNormalization(axis = 3)

        self.Conv2d2 = Conv2D(filters = self.F2, kernel_size = (self.kernel, self.kernel), strides = (1,1), padding = 'same',
                 kernel_initializer = glorot_uniform(seed=0))

        self.Conv2d3 = Conv2D(filters = self.F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid',
                 kernel_initializer = glorot_uniform(seed=0))

        self.Conv2d4 = Conv2D(filters = self.F3, kernel_size = (1, 1), strides = (self.stride,self.stride), padding = 'valid',
                        kernel_initializer = glorot_uniform(seed=0))


    def call(self, X,training=True):

      X_parallel = X

      X = self.Conv2d1(X)
      X = self.batch1(X)
      X = tf.nn.relu(X)


      X = self.Conv2d2(X)
      X = self.batch2(X)
      X = tf.nn.relu(X)



      X = self.Conv2d3(X)
      X = self.batch3(X)


      X_parallel = self.Conv2d4(X_parallel)
      X_parallel = self.batch4(X_parallel)


      X = Add()([X, X_parallel])
      X = tf.nn.relu(X)


      return X
```

In [185…
```
class identity_block(tf.keras.layers.Layer):
    def __init__(self, kernel=3,  filters=[4,4,8], name="identity_block"):
        super().__init__(name=name)
        self.F1, self.F2, self.F3 = filters
        self.kernel = kernel
        self.conv1 = Conv2D(filters = self.F1, kernel_size = (1, 1), strides = (1,1), padding = 'valid',
                 kernel_initializer = glorot_uniform(seed=0))
        self.batch1 = BatchNormalization(axis = 3, momentum=0.99, epsilon=0.001)

        self.conv2 = Conv2D(filters = self.F2, kernel_size = (self.kernel, self.kernel), strides = (1,1), padding = 'same',
                 kernel_initializer = glorot_uniform(seed=0))
        self.batch2 = BatchNormalization(axis = 3, momentum=0.99, epsilon=0.001)


        self.conv3 = Conv2D(filters = self.F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid',
```

```
                kernel_initializer = glorot_uniform(seed=0))
        self.batch3 = BatchNormalization(axis = 3, momentum=0.99, epsilon=0.001)


    def call(self, X,training=True):
      X_parallel = X

      X = self.conv1(X)
      X = self.batch1(X)
      X = Activation('relu')(X)


      X = self.conv2(X)
      X = self.batch2(X)
      X = Activation('relu')(X)


      X = self.conv3(X)
      X = self.batch3(X)


      X = Add()([X, X_parallel])
      X = Activation('relu')(X)

      return X
```

- The output of the C_4 will be passed to $\text{Chained Context Aggregation Module (CAM)}$



- The CAM module will have two operations names Context flow and Global flow
- **The Global flow**:
  - as shown in the above figure first we willl apply global avg pooling which results in (#, 1, 1, number_of_filters) then applying BN, RELU, $1 * 1$ Conv layer sequentially which results a matrix (#, 1, 1, number_of_filters). Finally apply upsampling / conv2d transpose to make the output same as the input dimensions (#, input_height, input_width, number_of_filters)
  - If you use upsampling then use bilinear pooling as interpolation technique
- **The Context flow**:
  - as shown in the above figure (c) the context flow will get inputs from two modules `a. C4  b. From the above flow`
  - We will be concatinating the both inputs on the last axis.
  - After the concatination we will be applying Average pooling which reduces the size of feature map by $N\times$ times
  - In the paper it was mentioned that to apply a group convolutions, but for the assignment we will be applying the simple conv layers with kernel size $(3 * 3)$
  - We are skipping the channel shuffling
  - similarly we will be applying a simple conv layers with kernel size $(3 * 3)$ consider this output is X
  - later we will get the Y=(X $\otimes \sigma((1 \times 1)conv(relu((1 \times 1)conv(X))))) \oplus X$, here $\oplus$ is elementwise addition and $\otimes$ is elementwise multiplication
  - Finally apply upsampling / conv2d transpose to make the output same as the input dimensions (#, input_height, input_width, number_of_filters)
  - If you use upsampling then use bilinear pooling as interpolation technique

NOTE: here N times reduction and N time increments makes the input and out shape same, you can explore with the N values, you can choose N = 2 or 4

- Example with N=2:
  - Assume the C4 is of shape (64,64,64) then the shape of GF will be (64,64,32)
  - Assume the C4 is of shape (64,64,64) and the shape of GF is (64,64,32) then the shape of CF1 will be (64,64,32)
  - Assume the C4 is of shape (64,64,64) and the shape of CF1 is (64,64,32) then the shape of CF2 will be (64,64,32)
  - Assume the C4 is of shape (64,64,64) and the shape of CF2 is (64,64,32) then the shape of CF3 will be (64,64,32)

In [186…

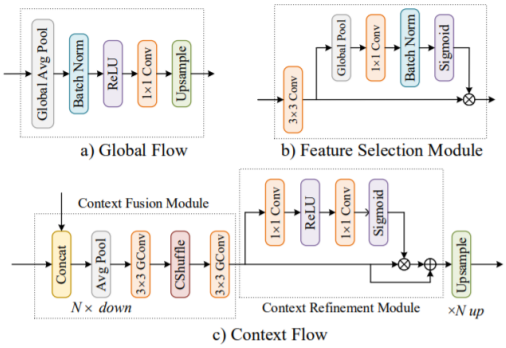
```
class global_flow(tf.keras.layers.Layer):
    def __init__(self, name="global_flow"):
        super().__init__(name=name)
        self.avg = tf.keras.layers.AveragePooling2D(strides= 1,pool_size = (16,16))
        self.batch = BatchNormalization()
        self.convd1 = Conv2D(filters = 32, kernel_size = (1, 1), activation = 'relu', padding = 'same')
        self.conT = (Conv2DTranspose(filters = 32 ,kernel_size = (16,16), use_bias = False))

    def call(self, X):
```

```
        output = X
        gf = self.avg(output)
        gf = self.batch(gf)
        gf = tf.nn.relu(gf)
        gf = self.convd1(gf)
        gf = self.conT(gf)
        return gf
```

In [210…

```
class context_flow(tf.keras.layers.Layer):
    def __init__(self, name="context_flow"):
        super().__init__(name=name)

        self.avg = AveragePooling2D((2, 2),  (2, 2), name = 'cf1_pool' )
        self.conv1 = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')
        self.conv2 = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')
        self.conv3 = Conv2D(filters = 32, kernel_size = (1, 1), activation = 'relu', padding = 'same')
        self.conv4 = Conv2D(filters = 32, kernel_size = (1, 1), activation = 'relu', padding = 'same')
        self.con_tra =  Conv2DTranspose(32 ,  (9, 9), use_bias = False )

    def call(self, X):
      # here X will a list of two elements
      INP, FLOW = X[0], X[1]
      Concat = Concatenate()([INP,FLOW])
      cf = self.avg(Concat)
      cf_1 = self.conv1(cf)
      cf_2 = self.conv2(cf_1)
      cf_3 = self.conv3(cf_2)
      cf_A = Activation('relu')(cf_3)
      cf_4 = self.conv4(cf_A)
      cf_A1 = Activation('sigmoid')(cf_4)
      cf_M = Multiply()([cf_2 , cf_A1])
      cf_add = Add()([cf_2,cf_M])
      cf_Tr = self.con_tra(cf_add)



        # implement the context flow as mentioned in the above cell
      return cf_Tr
```

- As shown in the above architecture we will be having 4 context flows
- if you have implemented correctly all the shapes of Global Flow, and 3 context flows will have the same dimension



- the output of these 4 modules will be added to get the same output matrix
  - The output of after the sum, will be sent to the **Feature selection module** FSM
- Example:
  - if the shapes of GF, CF1, CF2, CF3 are (64,64,32), (64,64,32), (64,64,32), (64,64,32), (64,64,32) respectivly then after the sum we will be getting (64,64,32), which will be passed to the next module.

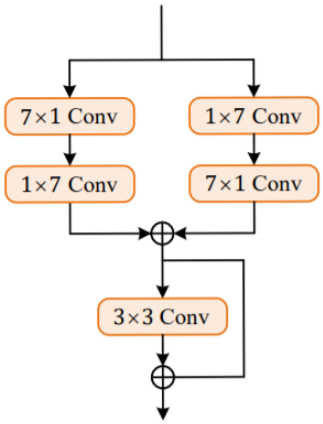
Feature selection module:

- As part of the FSM we will be applying a conv layer (3,3) with the padding="same" so that the output and input will have same shapes
- Let call the output as X
- Pass the X to global pooling which results the matrix (#, 1, 1, number_of_channels)
- Apply $1 * 1$ conv layer, after the pooling
- the output of the $1 * 1$ conv layer will be passed to the Batch normalization layer, followed by Sigmoid activation function.
- we will be having the output matrix of shape (#, 1, 1, number_of_channels) lets call it 'Y'
- **we can interpret this as attention mechanisum, i.e for each channel we will having a weight**
- the dimension of X (#, w, h, k) and output above steps Y is (#, 1, 1, k) i.e we need to multiply each channel of X will be multiplied with corresponding channel of Y
- After creating the weighted channel map we will be doing upsampling such that it will double the height and width.
- apply upsampling with bilinear pooling as interpolation technique

- Example:

- Assume the matrix shape of the input is (64,64,32) then after upsampling it will be (128,128,32)

In [212…
```
class fsm(tf.keras.layers.Layer):
    def __init__(self, name="feature_selection"):
        super().__init__(name=name)
        self.conv1 = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')
        self.avg = AveragePooling2D(1,(2, 2)) #check
        self.conv2 = Conv2D(filters = 32, kernel_size = (1, 1), activation = 'relu', padding = 'same')
        self.batch = BatchNormalization()
        self.contra = Conv2DTranspose(32 ,  (9, 9), use_bias = False )

    def call(self, X,training=True):
      FSM_1 = self.conv1(X)
      FSM_avg = self.avg(FSM_1) #check
      FSM = self.conv2(FSM_avg)
      FSM_bn = self.batch(FSM)
      FSM_s = Activation('sigmoid')(FSM_bn)
      FSM_T  = self.contra(FSM_s)
      FSM_T = Multiply()([FSM_1,FSM_T])

      # implement the FSM modules based on image in the above cells
      return FSM_T
```



b) AGCN

- **Adapted Global Convolutional Network (AGCN)**:
  - AGCN will get the input from the output of the "conv block" of C_1
  - In all the above layers we will be using the padding="same" and stride=(1,1)
  - so that we can have the input and output matrices of same size
- Example:
  - Assume the matrix shape of the input is (128,128,32) then the output it will be (128,128,32)

In [218…
```
class agcn(tf.keras.layers.Layer):
    def __init__(self, name="global_conv_net"):
        super().__init__(name=name)
        self.up =  UpSampling2D((2,2), interpolation='bilinear')
        self.conv1 = Conv2D(filters = 32, kernel_size = (7, 1), activation = 'relu', padding = 'same')
        self.conv2 = Conv2D(filters = 32, kernel_size = (1, 7), activation = 'relu', padding = 'same')

        self.conv3 = Conv2D(filters = 32, kernel_size = (1, 7), activation = 'relu', padding = 'same')
        self.conv4 = Conv2D(filters = 32, kernel_size = (7, 1), activation = 'relu', padding = 'same')

        self.conv5 = Conv2D(filters = 32, kernel_size =(3, 3), activation = 'relu', padding = 'same')
        self.up1 = UpSampling2D((2,2), interpolation='bilinear')

        self.conv6 = Conv2D(filters = 21, kernel_size = (3, 3), activation = 'relu', padding = 'same')
        self.conv7 = Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')

        self.up2  = UpSampling2D((4,4), interpolation = 'bilinear')


    def call(self, X,training=True):
      fsm, X_Stage_2_ = X[0], X[1]

      fsm =  self.conv7(fsm)

      ag_L1 = self.conv1(X_Stage_2_)
      ag_L2 = self.conv2(ag_L1)

      ag_L3 = self.conv3(X_Stage_2_)
```

```
        ag_L4 = self.conv4(ag_L3)

        sum1 = Add()([ag_L2,ag_L4])

        ag_conv = self.conv5(sum1)

        sum2 = Add()([sum1,ag_conv])



        fsm = self.up(fsm)
        sum2 = self.up1(sum2)
        CF = concatenate([fsm,sum2])
        X = self.conv6(CF)

        X  = self.up2(X)
        X = (Activation('softmax'))(X)
          # please implement the above mentioned architecture
        return X
```

In [223…

```
X_input = Input(shape=(128,128,3))

# Stage 1
X = Conv2D(64, (3, 3), name='conv1', padding="same", kernel_initializer=glorot_uniform(seed=0))(X_input)
X = BatchNormalization(axis=3, name='bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((2, 2), strides=(2, 2))(X)


print(X.shape)
```

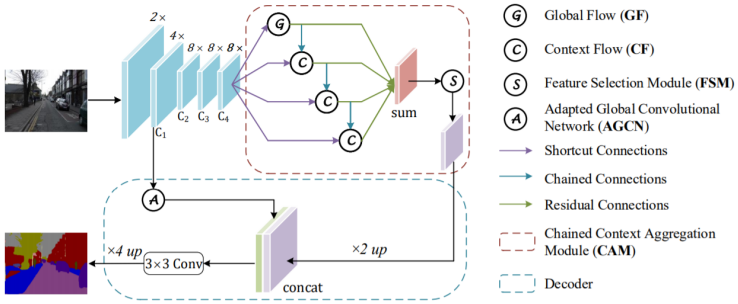
```
(None, 64, 64, 64)
```

In [224…

```
# Stage 2
convol1 = convolutional_block(kernel=3, filters=[4, 4, 8], stride=4,name = 'block_conv1' )
identity1 = identity_block( kernel = 3, filters = [4, 4, 8],name = 'identity1')
identity2 = identity_block( kernel = 3, filters = [4, 4, 8],name = 'identity2')
X_Stage_2_ = convol1(X)
X_Stage_2 = identity1(X_Stage_2_)
X_Stage_2 = identity2(X_Stage_2)


# Stage 3
convol2 = convolutional_block(kernel=3, filters=[8, 8, 16], stride=1,name = 'block_conv2')
identity3 = identity_block( kernel = 3, filters = [8, 8, 16],name = 'identity3')
identity4 = identity_block( kernel = 3, filters = [8, 8, 16],name = 'identity4')
identity5 = identity_block( kernel = 3, filters = [8, 8, 16],name = 'identity5')
X_Stage_3 = convol2(X_Stage_2)
X_Stage_3 = identity3(X_Stage_3)
X_Stage_3 = identity4(X_Stage_3)
X_Stage_3 = identity5(X_Stage_3)

# Stage 4
convol3 = convolutional_block(kernel=3, filters=[16, 16, 32], stride=1,name = 'block_conv3')
identity6 = identity_block( kernel = 3, filters = [16, 16, 32],name = 'identity6')
identity7 = identity_block( kernel = 3, filters = [16, 16, 32],name = 'identity7')
identity8 = identity_block( kernel = 3, filters = [16, 16, 32],name = 'identity8')
identity9 = identity_block( kernel = 3, filters = [16, 16, 32],name = 'identity9')
identity10 = identity_block( kernel = 3, filters = [16, 16, 32],name = 'identity10')
X_Stage_4 = convol3(X_Stage_3)
X_Stage_4 = identity6(X_Stage_4)
X_Stage_4 = identity7(X_Stage_4)
X_Stage_4 = identity8(X_Stage_4)
X_Stage_4 = identity9(X_Stage_4)
X_Stage_4 = identity10(X_Stage_4)

# Stage 5
convol4 = convolutional_block(kernel=3, filters=[32, 32, 64], stride=1,name = 'block_conv4')
identity11 = identity_block( kernel = 3, filters = [32, 32, 64],name = 'identity11')
identity12 = identity_block( kernel = 3, filters = [32, 32, 64],name = 'identity12')
X_Stage_5 = convol4(X_Stage_4)
X_Stage_5 = identity11(X_Stage_5)
X_Stage_5 = identity12(X_Stage_5)

#global flow
glo = global_flow()
gf = glo(X_Stage_5)

#Context flow
context2 = context_flow(name = 'cf1_pool2')
context3 = context_flow(name = 'cf1_pool3')
context4 = context_flow(name = 'cf1_pool4')

X1 = [X_Stage_5,gf]
cf_1 = context2(X1)

X1 = [X_Stage_5,cf_1]
cf_2 = context3(X1)

X1 = [X_Stage_5,cf_2]
cf_3 = context4(X1)
```

```
#adding
sum = Add(name = 'add')([gf, cf_1, cf_2, cf_3])

#feature selection module
feature = fsm()
FSM_T = feature(sum)


#Adapted Global Convolutional Network
Adapted = agcn()
X2 = [FSM_T, X_Stage_2_]

output = Adapted(X2)

model = Model(inputs = X_input, outputs = output)

for layer in model.layers:
  layer.trainable = True

model.summary()
```

```
Model: "model_19"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_18 (InputLayer)           [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 128, 128, 64) 1792        input_18[0][0]
__________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 128, 128, 64) 256         conv1[0][0]
__________________________________________________________________________________________________
activation_122 (Activation)     (None, 128, 128, 64) 0           bn_conv1[0][0]
__________________________________________________________________________________________________
max_pooling2d_16 (MaxPooling2D) (None, 64, 64, 64)   0           activation_122[0][0]
__________________________________________________________________________________________________
block_conv1 (convolutional_bloc (None, 16, 16, 8)    1064        max_pooling2d_16[0][0]
__________________________________________________________________________________________________
identity1 (identity_block)      (None, 16, 16, 8)    288         block_conv1[0][0]
__________________________________________________________________________________________________
identity2 (identity_block)      (None, 16, 16, 8)    288         identity1[0][0]
__________________________________________________________________________________________________
block_conv2 (convolutional_bloc (None, 16, 16, 16)   1136        identity2[0][0]
__________________________________________________________________________________________________
identity3 (identity_block)      (None, 16, 16, 16)   992         block_conv2[0][0]
__________________________________________________________________________________________________
identity4 (identity_block)      (None, 16, 16, 16)   992         identity3[0][0]
__________________________________________________________________________________________________
identity5 (identity_block)      (None, 16, 16, 16)   992         identity4[0][0]
__________________________________________________________________________________________________
block_conv3 (convolutional_bloc (None, 16, 16, 32)   4064        identity5[0][0]
__________________________________________________________________________________________________
identity6 (identity_block)      (None, 16, 16, 32)   3648        block_conv3[0][0]
__________________________________________________________________________________________________
identity7 (identity_block)      (None, 16, 16, 32)   3648        identity6[0][0]
__________________________________________________________________________________________________
identity8 (identity_block)      (None, 16, 16, 32)   3648        identity7[0][0]
__________________________________________________________________________________________________
identity9 (identity_block)      (None, 16, 16, 32)   3648        identity8[0][0]
__________________________________________________________________________________________________
identity10 (identity_block)     (None, 16, 16, 32)   3648        identity9[0][0]
__________________________________________________________________________________________________
block_conv4 (convolutional_bloc (None, 16, 16, 64)   15296       identity10[0][0]
__________________________________________________________________________________________________
identity11 (identity_block)     (None, 16, 16, 64)   13952       block_conv4[0][0]
__________________________________________________________________________________________________
identity12 (identity_block)     (None, 16, 16, 64)   13952       identity11[0][0]
__________________________________________________________________________________________________
global_flow (global_flow)       (None, 16, 16, 32)   264480      identity12[0][0]
__________________________________________________________________________________________________
cf1_pool2 (context_flow)        (None, 16, 16, 32)   121984      identity12[0][0]
                                                                 global_flow[0][0]
__________________________________________________________________________________________________
cf1_pool3 (context_flow)        (None, 16, 16, 32)   121984      identity12[0][0]
                                                                 cf1_pool2[0][0]
__________________________________________________________________________________________________
cf1_pool4 (context_flow)        (None, 16, 16, 32)   121984      identity12[0][0]
                                                                 cf1_pool3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 16, 16, 32)   0           global_flow[0][0]
                                                                 cf1_pool2[0][0]
                                                                 cf1_pool3[0][0]
                                                                 cf1_pool4[0][0]
__________________________________________________________________________________________________
feature_selection (fsm)         (None, 16, 16, 32)   93376       add[0][0]
__________________________________________________________________________________________________
global_conv_net (agcn)          (None, 128, 128, 21) 48661       feature_selection[0][0]
                                                                 block_conv1[0][0]
==================================================================================================
Total params: 845,773
Trainable params: 843,325
Non-trainable params: 2,448
__________________________________________________________________________________________________
```



- as shown in the architecture, after we get the AGCN it will get concatinated with the FSM output
- If we observe the shapes both AGCN and FSM will have same height and weight
- we will be concatinating both these outputs over the last axis
- The concatinated output will be passed to a conv layers with filters = number of classes in our data set and the activation function = 'relu'
- we will be using padding="same" which results in the same size feature map
- If you observe the shape of matrix, it will be 4x times less than the original image
- to make it equal to the original output shape, we will do 4x times upsampling of rows and columns
- apply upsampling with bilinear pooling as interpolation technique
- Finally we will be applying sigmoid activation.
- Example:
  - Assume the matrix shape of AGCN is (128,128,32) and FSM is (128,128,32) the concatination will make it (128, 128, 64)
  - Applying conv layer will make it (128,128,21)
  - Finally applying upsampling will make it (512, 512, 21)
  - Applying sigmoid will result in the same matrix (512, 512, 21)
- If you observe the arcitecture we are creating a feature map with 2x time less width and height
- we have written the first stage of the code above.
- Write the next layers by using the custom layers we have written

In [ ]:
```
rms = tf.keras.optimizers.RMSprop(learning_rate=0.0001)
focal_loss = sm.losses.cce_dice_loss
model.compile(rms, focal_loss, metrics=[iou_score])
```

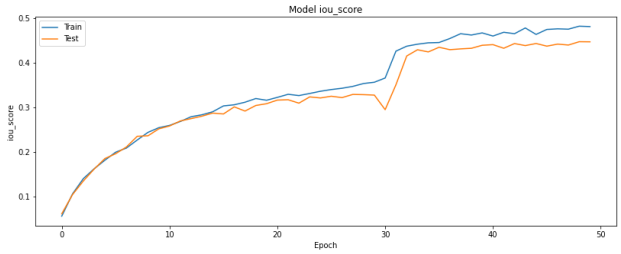
In [30]:
```
callbacks_ = [
    tf.keras.callbacks.ModelCheckpoint('./best_model2.h5', save_weights_only=True, save_best_only=True, \
                                       mode='min', monitor='val_iou_score')]

history = model.fit(train_dataloader, steps_per_epoch=len(train_dataloader)//5, epochs=50,\
                            validation_data=test_dataloader,validation_steps =len(test_dataloader)//5,callbacks=callbacks_ )
```

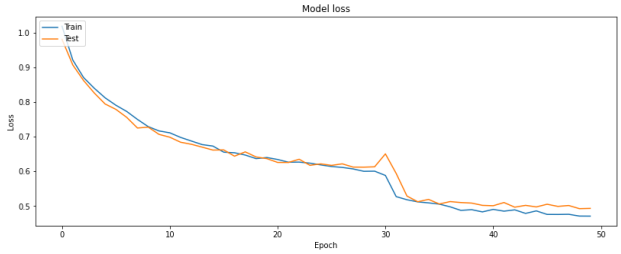
```
Epoch 1/50
136/136 [==============================] - 653s 5s/step - loss: 1.0184 - iou_score: 0.0557 - val_loss: 0.9806 - val_iou_score: 0.0610
Epoch 2/50
136/136 [==============================] - 482s 4s/step - loss: 0.9209 - iou_score: 0.1056 - val_loss: 0.9067 - val_iou_score: 0.1040
Epoch 3/50
136/136 [==============================] - 428s 3s/step - loss: 0.8705 - iou_score: 0.1398 - val_loss: 0.8618 - val_iou_score: 0.1344
Epoch 4/50
136/136 [==============================] - 395s 3s/step - loss: 0.8394 - iou_score: 0.1618 - val_loss: 0.8257 - val_iou_score: 0.1612
Epoch 5/50
136/136 [==============================] - 376s 3s/step - loss: 0.8119 - iou_score: 0.1811 - val_loss: 0.7942 - val_iou_score: 0.1848
Epoch 6/50
136/136 [==============================] - 350s 3s/step - loss: 0.7906 - iou_score: 0.1991 - val_loss: 0.7787 - val_iou_score: 0.1956
Epoch 7/50
136/136 [==============================] - 301s 2s/step - loss: 0.7727 - iou_score: 0.2084 - val_loss: 0.7557 - val_iou_score: 0.2108
Epoch 8/50
136/136 [==============================] - 307s 2s/step - loss: 0.7497 - iou_score: 0.2264 - val_loss: 0.7249 - val_iou_score: 0.2347
Epoch 9/50
136/136 [==============================] - 284s 2s/step - loss: 0.7289 - iou_score: 0.2436 - val_loss: 0.7275 - val_iou_score: 0.2359
Epoch 10/50
136/136 [==============================] - 290s 2s/step - loss: 0.7166 - iou_score: 0.2539 - val_loss: 0.7066 - val_iou_score: 0.2514
Epoch 11/50
136/136 [==============================] - 273s 2s/step - loss: 0.7111 - iou_score: 0.2591 - val_loss: 0.6984 - val_iou_score: 0.2578
Epoch 12/50
136/136 [==============================] - 269s 2s/step - loss: 0.6979 - iou_score: 0.2679 - val_loss: 0.6840 - val_iou_score: 0.2691
Epoch 13/50
136/136 [==============================] - 259s 2s/step - loss: 0.6875 - iou_score: 0.2786 - val_loss: 0.6781 - val_iou_score: 0.2743
Epoch 14/50
136/136 [==============================] - 258s 2s/step - loss: 0.6771 - iou_score: 0.2829 - val_loss: 0.6698 - val_iou_score: 0.2798
Epoch 15/50
136/136 [==============================] - 259s 2s/step - loss: 0.6729 - iou_score: 0.2896 - val_loss: 0.6614 - val_iou_score: 0.2867
```

```
Epoch 16/50
136/136 [==============================] - 262s 2s/step - loss: 0.6554 - iou_score: 0.3028 - val_loss: 0.6619 - val_iou_score: 0.2848
Epoch 17/50
136/136 [==============================] - 259s 2s/step - loss: 0.6536 - iou_score: 0.3053 - val_loss: 0.6438 - val_iou_score: 0.3007
Epoch 18/50
136/136 [==============================] - 255s 2s/step - loss: 0.6470 - iou_score: 0.3111 - val_loss: 0.6562 - val_iou_score: 0.2913
Epoch 19/50
136/136 [==============================] - 251s 2s/step - loss: 0.6368 - iou_score: 0.3194 - val_loss: 0.6416 - val_iou_score: 0.3038
Epoch 20/50
136/136 [==============================] - 252s 2s/step - loss: 0.6399 - iou_score: 0.3155 - val_loss: 0.6368 - val_iou_score: 0.3078
Epoch 21/50
136/136 [==============================] - 248s 2s/step - loss: 0.6343 - iou_score: 0.3220 - val_loss: 0.6258 - val_iou_score: 0.3159
Epoch 22/50
136/136 [==============================] - 245s 2s/step - loss: 0.6264 - iou_score: 0.3289 - val_loss: 0.6257 - val_iou_score: 0.3167
Epoch 23/50
136/136 [==============================] - 248s 2s/step - loss: 0.6267 - iou_score: 0.3260 - val_loss: 0.6350 - val_iou_score: 0.3090
Epoch 24/50
136/136 [==============================] - 243s 2s/step - loss: 0.6236 - iou_score: 0.3307 - val_loss: 0.6174 - val_iou_score: 0.3230
Epoch 25/50
136/136 [==============================] - 246s 2s/step - loss: 0.6186 - iou_score: 0.3357 - val_loss: 0.6216 - val_iou_score: 0.3207
Epoch 26/50
136/136 [==============================] - 244s 2s/step - loss: 0.6140 - iou_score: 0.3395 - val_loss: 0.6173 - val_iou_score: 0.3245
Epoch 27/50
136/136 [==============================] - 244s 2s/step - loss: 0.6114 - iou_score: 0.3424 - val_loss: 0.6217 - val_iou_score: 0.3214
Epoch 28/50
136/136 [==============================] - 247s 2s/step - loss: 0.6070 - iou_score: 0.3466 - val_loss: 0.6123 - val_iou_score: 0.3286
Epoch 29/50
136/136 [==============================] - 245s 2s/step - loss: 0.6002 - iou_score: 0.3531 - val_loss: 0.6121 - val_iou_score: 0.3281
Epoch 30/50
136/136 [==============================] - 245s 2s/step - loss: 0.6006 - iou_score: 0.3560 - val_loss: 0.6131 - val_iou_score: 0.3271
Epoch 31/50
136/136 [==============================] - 243s 2s/step - loss: 0.5882 - iou_score: 0.3654 - val_loss: 0.6504 - val_iou_score: 0.2944
Epoch 32/50
136/136 [==============================] - 244s 2s/step - loss: 0.5272 - iou_score: 0.4256 - val_loss: 0.5940 - val_iou_score: 0.3502
Epoch 33/50
136/136 [==============================] - 245s 2s/step - loss: 0.5182 - iou_score: 0.4367 - val_loss: 0.5288 - val_iou_score: 0.4147
Epoch 34/50
136/136 [==============================] - 244s 2s/step - loss: 0.5121 - iou_score: 0.4412 - val_loss: 0.5124 - val_iou_score: 0.4287
Epoch 35/50
136/136 [==============================] - 246s 2s/step - loss: 0.5092 - iou_score: 0.4443 - val_loss: 0.5193 - val_iou_score: 0.4238
Epoch 36/50
136/136 [==============================] - 246s 2s/step - loss: 0.5056 - iou_score: 0.4450 - val_loss: 0.5060 - val_iou_score: 0.4343
Epoch 37/50
136/136 [==============================] - 246s 2s/step - loss: 0.4978 - iou_score: 0.4539 - val_loss: 0.5129 - val_iou_score: 0.4289
Epoch 38/50
136/136 [==============================] - 246s 2s/step - loss: 0.4875 - iou_score: 0.4647 - val_loss: 0.5102 - val_iou_score: 0.4307
Epoch 39/50
136/136 [==============================] - 245s 2s/step - loss: 0.4896 - iou_score: 0.4617 - val_loss: 0.5086 - val_iou_score: 0.4320
Epoch 40/50
136/136 [==============================] - 245s 2s/step - loss: 0.4832 - iou_score: 0.4665 - val_loss: 0.5021 - val_iou_score: 0.4388
Epoch 41/50
136/136 [==============================] - 246s 2s/step - loss: 0.4904 - iou_score: 0.4595 - val_loss: 0.5010 - val_iou_score: 0.4402
Epoch 42/50
136/136 [==============================] - 247s 2s/step - loss: 0.4853 - iou_score: 0.4679 - val_loss: 0.5102 - val_iou_score: 0.4321
Epoch 43/50
136/136 [==============================] - 246s 2s/step - loss: 0.4891 - iou_score: 0.4647 - val_loss: 0.4968 - val_iou_score: 0.4426
Epoch 44/50
136/136 [==============================] - 247s 2s/step - loss: 0.4788 - iou_score: 0.4776 - val_loss: 0.5021 - val_iou_score: 0.4381
Epoch 45/50
136/136 [==============================] - 246s 2s/step - loss: 0.4861 - iou_score: 0.4631 - val_loss: 0.4975 - val_iou_score: 0.4427
Epoch 46/50
136/136 [==============================] - 247s 2s/step - loss: 0.4759 - iou_score: 0.4743 - val_loss: 0.5052 - val_iou_score: 0.4368
Epoch 47/50
136/136 [==============================] - 247s 2s/step - loss: 0.4758 - iou_score: 0.4756 - val_loss: 0.4990 - val_iou_score: 0.4414
Epoch 48/50
136/136 [==============================] - 244s 2s/step - loss: 0.4763 - iou_score: 0.4749 - val_loss: 0.5016 - val_iou_score: 0.4393
Epoch 49/50
136/136 [==============================] - 245s 2s/step - loss: 0.4711 - iou_score: 0.4816 - val_loss: 0.4925 - val_iou_score: 0.4467
Epoch 50/50
136/136 [==============================] - 247s 2s/step - loss: 0.4710 - iou_score: 0.4807 - val_loss: 0.4933 - val_iou_score: 0.4465
```

In [31]:

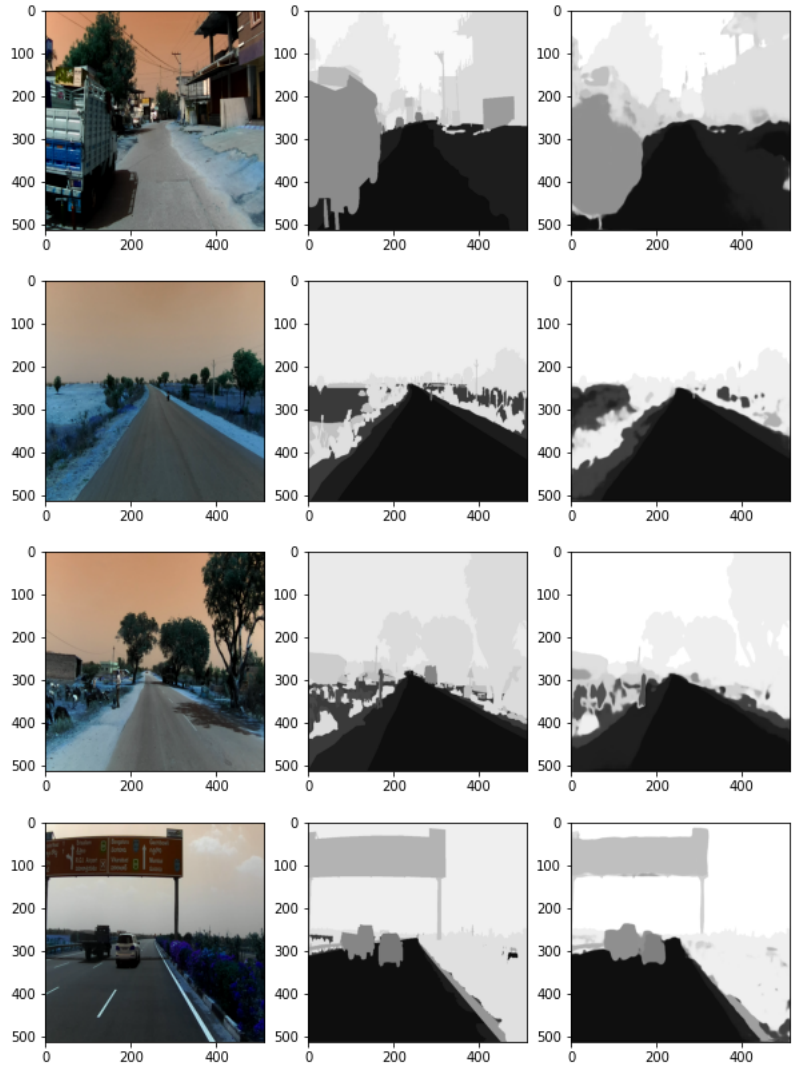
```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['iou_score'])
plt.plot(history.history['val_iou_score'])
plt.title('Model iou_score')
plt.ylabel('iou_score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

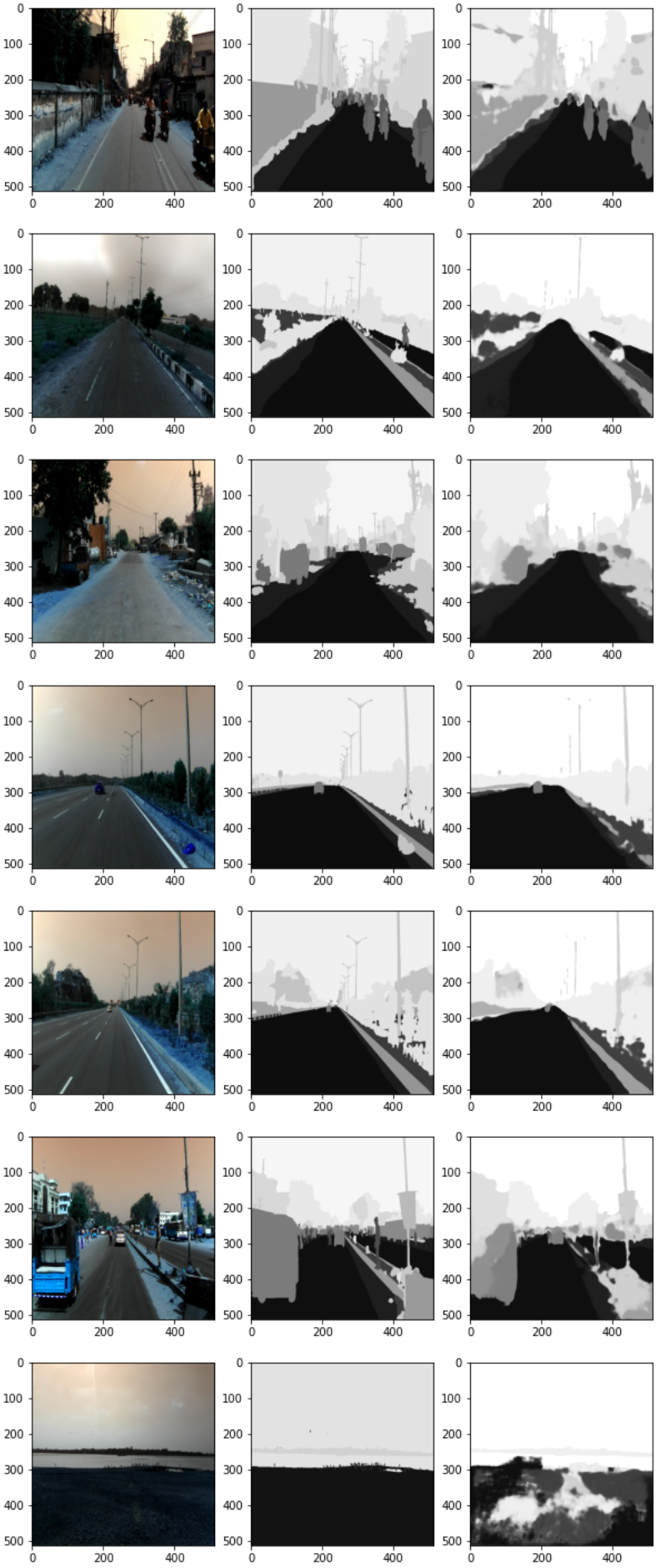


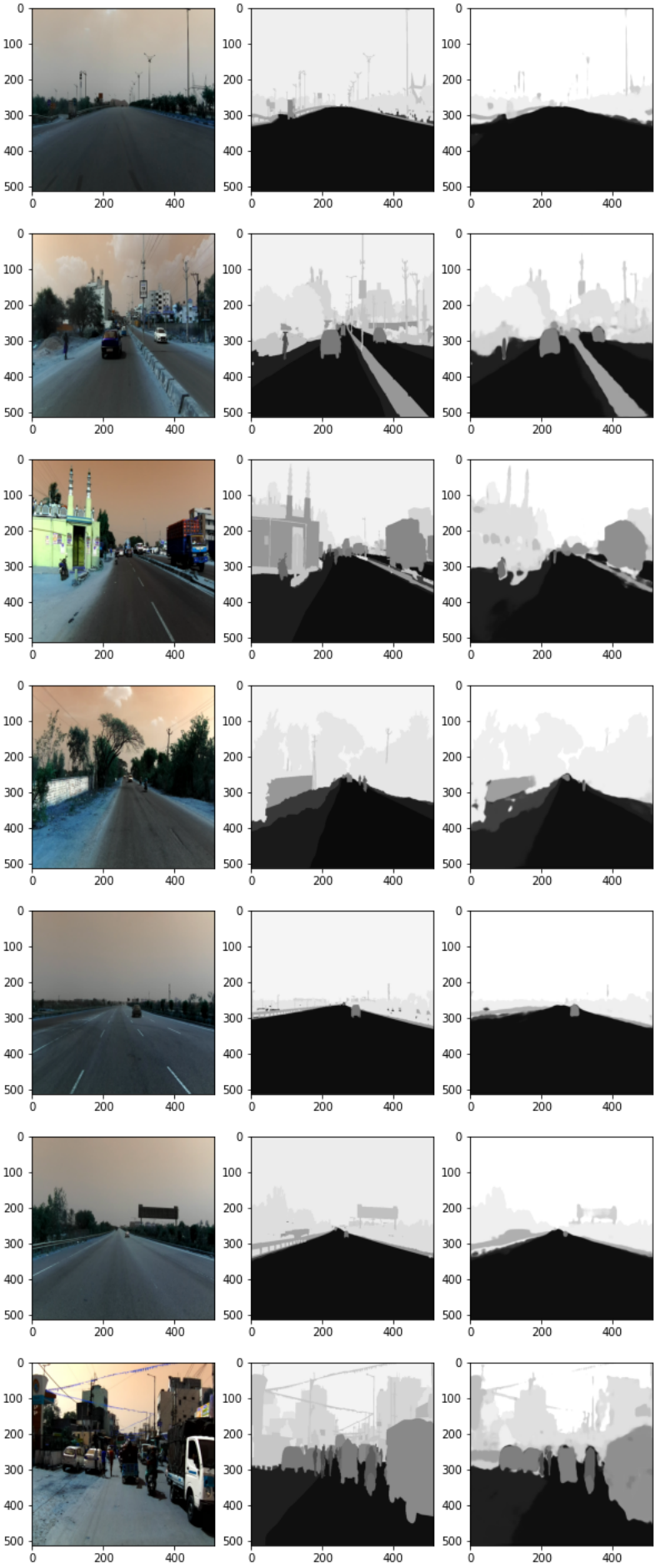


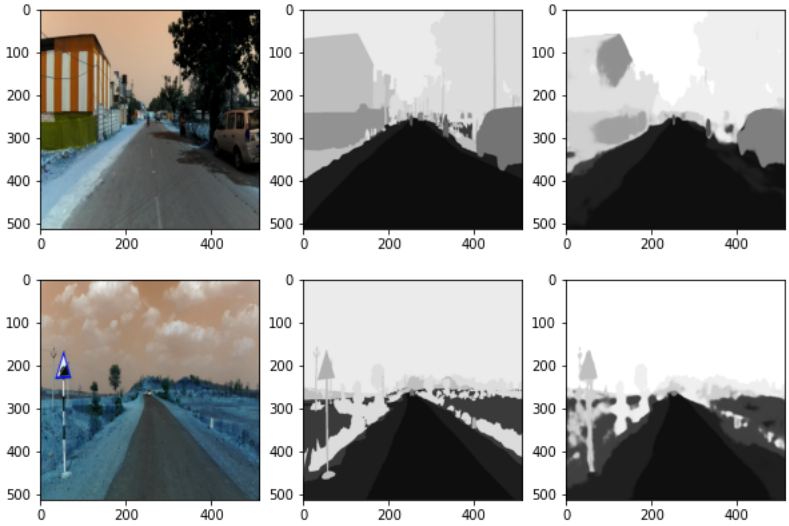
In [149…

```
a = X_test
for i in range(20):
    #original image
    image = cv2.imread(a[i][0],cv2.IMREAD_UNCHANGED)
    image = cv2.resize(image, (512, 512))

    #predicted segmentation map
    predicted  = model.predict(image[np.newaxis,:,:,:])
    predict = np.array([predicted[0,:,:,i]*j for i , j in enumerate(class_label)])
    predict = sum(predict)

    #original segmentation map
    image_mask = cv2.imread(a[i][1], cv2.IMREAD_UNCHANGED)
    image_mask = cv2.resize(image_mask, (512,512))


    plt.figure(figsize=(10,6))
    plt.subplot(131)
    plt.imshow(image)
    plt.subplot(132)
    plt.imshow(image_mask, cmap='gray')
    plt.subplot(133)
    plt.imshow(predict, cmap='gray')
    plt.show()
```









Usefull tips:

- use "interpolation=cv2.INTER_NEAREST" when you are resizing the image, so that it won't mess with the number of classes
- keep the images in the square shape like $256 * 256$ or $512 * 512$
- Carefull when you are converting the (W, H) output image into (W, H, Classes)
- Even for the canet, use the segmentation model's losses and the metrics
- The goal of this assignment is make you familier in with computer vision problems, image preprocessing, building complex architectures and implementing research papers, so that in future you will be very confident in industry
- you can use the tensorboard logss to see how is yours model's training happening
- use callbacks that you have implemented in previous assignments

Things to keep in mind

- You need to train above built model and plot the train and test losses.
- Make sure there is no overfitting, you are free play with the identity blocks in C1, C2, C3, C4
- before we apply the final sigmoid activation, you can add more conv layers or BN or dropouts etc
- you are free to use any other optimizer or learning rate or weights init or regularizations