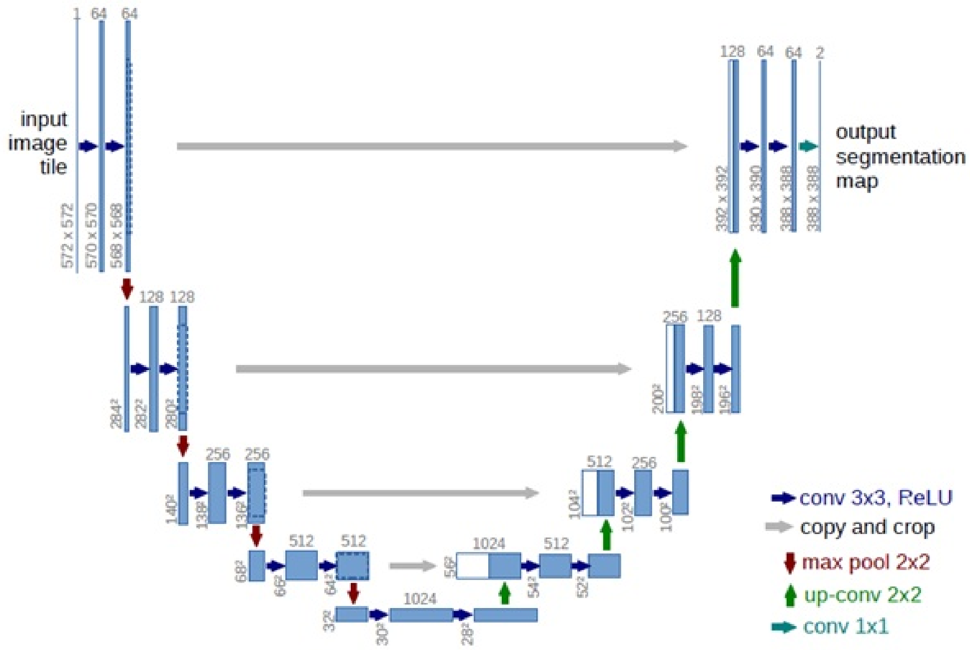
U-Net for Semantic Segmentation on Indian Driving Dataset

U-Net is a convolutional neural network that was developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg, Germany, The network is based on the fully convolutional network and its architecture is designed to work with fewer training images and to yield more precise segmentation.The below image represents the U-Net architecture



Implementation Detail:

The Blue box represents the feature maps that are obtained after an operation is applied and all operations are represented by different color arrow marks, The numbers of filters or depth of the feature maps are represented at the top of the blue box, resolution of the feature map is represented at the bottom left of each blue box, the White box is the feature maps that are copied from the adjacent block for concatenation for further operation.

- **The implementation consists of a contracting path and an expansive path.**
- **The contracting path (Encoder) follows the typical architecture of a convolutional network, It consists of the repeated two 3x3 unpadded convolutions followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling.**
- **The Encoder is for downsampling where the number of feature map channels is doubled for each successive block.**
- **The expansive path (Decoder) consists of 2x2 up-convolution that is applied to feature maps that have a cropped feature concatenated with it from contraction path followed by two 3x3 convolutions, each with ReLU.**
- **The Decoder is for upsampling where the number of feature map channels is reduced by a factor of 2 for each successive block.**
- **The Final layer has a 1x1 convolution that is used to map each 64 component feature vector to the desired number of classes and network has a total of 23 convolutional layers.**
- **For more information please refer the paper U-Net Convolutional Networks for Biomedical Image Segmentation and its summary here**

Installing required modules

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from numpy import asarray,zeros,moveaxis
from tensorflow.keras.initializers import *
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.optimizers import *
import matplotlib.pyplot as plt
from sys import getsizeof
from tensorflow.keras.callbacks import TensorBoard
from tqdm import tqdm_notebook,tqdm
from sklearn.metrics import *
import os,sys,ntpath,fnmatch,shutil,cv2
import joblib,os.path,itertools,warnings
```

```
from tensorflow.keras.models import load_model
from scipy.sparse import csc_matrix
import numpy as np
import pandas as pd
from os import path
!pip install import_ipynb
from IPython.display import clear_output
from time import time
np.random.seed(0)
from google.colab import drive
drive.mount('/content/drive')
! cp -v "/content/drive/My Drive/IID_Files/Utility_Functions.ipynb" "/content"
! cp -v "/content/drive/My Drive/IID_Files/Final.ipynb" "/content"
! cp -v "/content/drive/My Drive/IID_Files/IID_Data_Prep_Utils1.ipynb" "/content"
warnings.filterwarnings("ignore")
clear_output()
```

Importing Data Preparation Modules

In [ ]:

```
import import_ipynb
from Utility_Functions import *
from Final import *
from IID_Data_Prep_Utils1 import *
```

```
Checking Status:
-------------------------------------------------------
1.Image Data Preparation     .. .. .. >>> |Done| <1/5>
2.Label Mask Preparation     .. .. .. >>> |Done| <2/5>
3.Data Shuffling             .. .. .. >>> |Done| <3/5>
4.Data Train_Test_Split      .. .. .. >>> |Done| <4/5>
5.Loading Final Data         .. .. .. >>> |Done| <5/5>
-------------------------------------------------------

Gen RAM Free: 6.68 GB  - Used: 17.98 GB - Total : 25.51 GB - Util  70.48 %
GPU RAM Free: 15.9 GB  - Used: 0.0 GB - Total : 15.9 GB  - Util  0.0 %
```

General Utility Function for Prediction

In [ ]:

```
def predict_for(data_for_prediction, weights_save_path=False):

    """  General Function to perform prediction for the specified data split  """

    Mean_MIoU, Accuracy, cf_matrix=[], [], np.zeros((7,7))
    x, y = Load_For_Prediction(data_for_prediction)
    Model, Skip = Select_Model(weights_save_path), 2

    for d in tqdm_notebook(range(0,len(x),Skip)):

        if (d>=(len(x)-Skip)):
            plot,_,_=True,clear_output(),print("Total number of samples in {0} : {1}".format(data_for_pr
ediction,len(x)))
        else: plot=False

        Miou, cf_matrix, Accuracy=Function_2(x[d:d+Skip],y[d:d+Skip],Mean_MIoU,cf_matrix,Accuracy,Model,
plot,False,False,False)
    collected = gc.collect()

    return Miou, Accuracy, cf_matrix
```

Implementation of U-Net

In [ ]:

```
def Unet_Segmentation(input_shape, n_classes):

    """
    Function to build U-Net Architecture for Image Segmentation
    Input  : input_shape <tuple>, n_classes <Int>
    Return : Unet_model """

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

        """
```

```
        Function to Build U-Net Encoder Blocks
        Input  : Block_Number <Int>, Name <String>, Filters <Int>, Kernel_Size <Tuple>,  Pool_size <Tuple>, Previous_layer <Keras.layer>, initialize <String>
        Return : Convolution2 <Keras.layer> """

        # Defining Max-pooling layer for each Encoder Block
        MaxPool = tf.keras.layers.MaxPooling2D(pool_size=Pool_size, name= Name+"_Maxpool")(Previous_layer) if Block_Number>1 else Previous_layer

        # Defining two Convolution layers for each Encoder Block
        Convolution1 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv1", activation = 'relu', kernel_initializer= initialize, padding='same')(MaxPool)
        Convolution2 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv2", activation = 'relu', kernel_initializer= initialize, padding='same')(Convolution1)

        return Convolution2

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

        """
        Function to Build U-Net Decoder Blocks
        Input  : Block_Number <Int>, Name <String>, Filters <Int>, Kernel_Size <Tuple>, Previous_layer <Keras.layer>,  Layer_to_Concatenate <Keras.layer>,  initialize <String>
        Return : Convolution2 <Keras.layer> """


        # Defining Up-Convolution layers for Decoder Blocks
        Up_Sample = tf.keras.layers.UpSampling2D(size=(2, 2), name= Name+"_Upsample")(Previous_layer)
        Up_Convolution = tf.keras.layers.Conv2D(Filters,(2,2), name= Name+"_UpConv", activation = 'relu', padding = 'same',kernel_initializer=initialize)(Up_Sample)

        # concatenating feature maps that are copied from encoder block with Previous Layer
        Concatenated_Layer=tf.keras.layers.Concatenate(axis=3, name= Name+"_Concat")([Layer_to_Concatenate,Up_Convolution])

        # Defining two Convolution layers for each Decoder Block
        Convolution1 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv1", activation = 'relu', kernel_initializer= initialize, padding ='same')(Concatenated_Layer)
        Convolution2 = tf.keras.layers.Conv2D(Filters, Kernel_Size, name= Name+"_Conv2", activation = 'relu', kernel_initializer= initialize, padding ='same')(Convolution1)

        if Block_Number==4:

            # Final Convolution layer has number of classes as filter size followed by softmax
            Convolution3 = tf.keras.layers.Conv2D(n_classes, Kernel_Size, name= "Final_Conv", activation = 'relu', kernel_initializer= initialize, padding ='same')(Convolution2)
            Output=Activation('softmax', name="Softmax")(Convolution3)

            return Output

        return Convolution2

    # Input Layer of U-Net
    Input_layer = tf.keras.layers.Input(shape=input_shape)

    # Building Encoder Block for U-Net Various filters Sizes
    En_Block1 = Unet_En_Blocks(1, "En_Block1", 64,  (3,3), (2,2), Input_layer)
    En_Block2 = Unet_En_Blocks(2, "En_Block2", 128, (3,3), (2,2), En_Block1)
    En_Block3 = Unet_En_Blocks(3, "En_Block3", 256, (3,3), (2,2), En_Block2)
    En_Block4 = Unet_En_Blocks(4, "En_Block4", 512, (3,3), (2,2), En_Block3)
    En_Block5 = Unet_En_Blocks(5, "En_Block5", 1024, (3,3), (2,2), En_Block4)

    # Building Decoder Block for U-Net Various filters Sizes
    Dec_Block1 = Unet_Dec_Blocks(1, "Dec_Block1", 512,  (3,3), En_Block5, En_Block4)
    Dec_Block2 = Unet_Dec_Blocks(2, "Dec_Block2", 256, (3,3), Dec_Block1, En_Block3)
    Dec_Block3 = Unet_Dec_Blocks(3, "Dec_Block3", 128, (3,3), Dec_Block2, En_Block2)
    Output_layer = Unet_Dec_Blocks(4, "Dec_Block4", 64, (3,3), Dec_Block3, En_Block1)

    # Invoke Model to get U-Net model
    Unet_model = Model(Input_layer, Output_layer)

    return Unet_model

# Invoke Unet_Segmentation to get U-Net model
input_shape, n_classes = (240, 480,3), 7
Unet = Unet_Segmentation(input_shape, n_classes)
```

Training U-Net Model

In [ ]:

```
# Get current Time
start_time = time()

# Defining Batch size and epoch
batch_size, epochs = 16, 50

# Defining tensorboard to store Training Information and filepath to store Unet model
tensorboard, filepath = TensorBoard(log_dir=root+"logs/unet_{}".format(str(time())[5:10])),root+"/Unet.be
st.hdf5"

# Defining steps_per_epoch and validation_steps for Training
steps_per_epoch,validation_steps=int((len(train_img_files1)+len(train_img_files2))/batch_size),int((len(v
al_img_files1)+len(val_img_files2))/batch_size)

# Compile U-net Model
Unet.compile(optimizer = tf.keras.optimizers.Adam(0.0001), loss = 'categorical_crossentropy',metrics = ['
accuracy',miou])

# Defining EarlyStopping with patience=5 and monitor='val_miou'
es = EarlyStopping(monitor='val_miou', mode='max', verbose=1, patience=5)

# Defining ModelCheckpoint with monitor as 'val_miou'
checkpoint = ModelCheckpoint(filepath, monitor='val_miou', verbose=2, save_best_only=True, mode='max')

# Defining ReduceLROnPlateau to reduce learning rate with patience=3
learning_rate_reduction = ReduceLROnPlateau(monitor='val_miou', patience=3, verbose=2, factor=0.2, min_lr
=0.00001)

# Fit U-net Model to start training
history=Unet.fit_generator(train_batch_generator(batch_size,epochs), steps_per_epoch=steps_per_epoch, epoc
hs=epochs, verbose=1, validation_data=val_batch_generator(batch_size,epochs),
                           validation_steps=validation_steps, callbacks=[learning_rate_reduction,checkp
oint,es,tensorboard])

# Printing Time taken for Training
print("--- %s seconds ---" % (time() - start_time))
```

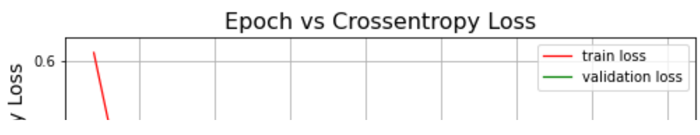
```
Epoch 1/50
876/876 [==============================] - ETA: 0s - loss: 0.6122 - accuracy: 0.7814 - miou: 0.4298
Epoch 00001: val_miou improved from -inf to 0.46940, saving model to /content/drive/My Drive/Unet.best.hdf
5
876/876 [==============================] - 1044s 1s/step - loss: 0.6122 - accuracy: 0.7814 - miou: 0.4298
- val_loss: 0.4783 - val_accuracy: 0.8184 - val_miou: 0.4694
Epoch 2/50
876/876 [==============================] - ETA: 0s - loss: 0.3988 - accuracy: 0.8465 - miou: 0.5583
Epoch 00002: val_miou improved from 0.46940 to 0.57656, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1064s 1s/step - loss: 0.3988 - accuracy: 0.8465 - miou: 0.5583
- val_loss: 0.3865 - val_accuracy: 0.8452 - val_miou: 0.5766
Epoch 3/50
876/876 [==============================] - ETA: 0s - loss: 0.3358 - accuracy: 0.8702 - miou: 0.6196
Epoch 00003: val_miou improved from 0.57656 to 0.59402, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1063s 1s/step - loss: 0.3358 - accuracy: 0.8702 - miou: 0.6196
- val_loss: 0.3589 - val_accuracy: 0.8619 - val_miou: 0.5940
Epoch 4/50
876/876 [==============================] - ETA: 0s - loss: 0.2971 - accuracy: 0.8848 - miou: 0.6549
Epoch 00004: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 00004: val_miou improved from 0.59402 to 0.61716, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1057s 1s/step - loss: 0.2971 - accuracy: 0.8848 - miou: 0.6549
- val_loss: 0.3384 - val_accuracy: 0.8693 - val_miou: 0.6172
Epoch 5/50
876/876 [==============================] - ETA: 0s - loss: 0.2496 - accuracy: 0.9029 - miou: 0.7008
Epoch 00005: val_miou improved from 0.61716 to 0.63516, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1061s 1s/step - loss: 0.2496 - accuracy: 0.9029 - miou: 0.7008
- val_loss: 0.3134 - val_accuracy: 0.8821 - val_miou: 0.6352
Epoch 6/50
876/876 [==============================] - ETA: 0s - loss: 0.2298 - accuracy: 0.9110 - miou: 0.7190
Epoch 00006: val_miou improved from 0.63516 to 0.64279, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1056s 1s/step - loss: 0.2298 - accuracy: 0.9110 - miou: 0.7190
- val_loss: 0.3180 - val_accuracy: 0.8814 - val_miou: 0.6428
Epoch 7/50
876/876 [==============================] - ETA: 0s - loss: 0.2142 - accuracy: 0.9175 - miou: 0.7355
Epoch 00007: ReduceLROnPlateau reducing learning rate to 1e-05.

Epoch 00007: val_miou improved from 0.64279 to 0.64376, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1049s 1s/step - loss: 0.2142 - accuracy: 0.9175 - miou: 0.7355
- val_loss: 0.3266 - val_accuracy: 0.8831 - val_miou: 0.6438
```

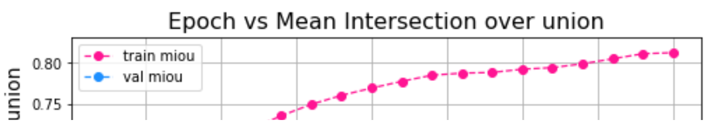
```
Epoch 8/50
876/876 [==============================] - ETA: 0s - loss: 0.1992 - accuracy: 0.9238 - miou: 0.7495
Epoch 00008: val_miou did not improve from 0.64376
876/876 [==============================] - 1052s 1s/step - loss: 0.1992 - accuracy: 0.9238 - miou: 0.7495
- val_loss: 0.3434 - val_accuracy: 0.8819 - val_miou: 0.6434
Epoch 9/50
876/876 [==============================] - ETA: 0s - loss: 0.1894 - accuracy: 0.9279 - miou: 0.7604
Epoch 00009: val_miou improved from 0.64376 to 0.64721, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1056s 1s/step - loss: 0.1894 - accuracy: 0.9279 - miou: 0.7604
- val_loss: 0.3581 - val_accuracy: 0.8819 - val_miou: 0.6472
Epoch 10/50
876/876 [==============================] - ETA: 0s - loss: 0.1808 - accuracy: 0.9314 - miou: 0.7694
Epoch 00010: val_miou did not improve from 0.64721
876/876 [==============================] - 1044s 1s/step - loss: 0.1808 - accuracy: 0.9314 - miou: 0.7694
- val_loss: 0.3594 - val_accuracy: 0.8839 - val_miou: 0.6416
Epoch 11/50
876/876 [==============================] - ETA: 0s - loss: 0.1726 - accuracy: 0.9347 - miou: 0.7776
Epoch 00011: val_miou did not improve from 0.64721
876/876 [==============================] - 1041s 1s/step - loss: 0.1726 - accuracy: 0.9347 - miou: 0.7776
- val_loss: 0.3717 - val_accuracy: 0.8828 - val_miou: 0.6438
Epoch 12/50
876/876 [==============================] - ETA: 0s - loss: 0.1670 - accuracy: 0.9369 - miou: 0.7851
Epoch 00012: val_miou did not improve from 0.64721
876/876 [==============================] - 1038s 1s/step - loss: 0.1670 - accuracy: 0.9369 - miou: 0.7851
- val_loss: 0.3820 - val_accuracy: 0.8832 - val_miou: 0.6421
Epoch 13/50
876/876 [==============================] - ETA: 0s - loss: 0.1645 - accuracy: 0.9378 - miou: 0.7877
Epoch 00013: val_miou did not improve from 0.64721
876/876 [==============================] - 1044s 1s/step - loss: 0.1645 - accuracy: 0.9378 - miou: 0.7877
- val_loss: 0.3796 - val_accuracy: 0.8829 - val_miou: 0.6449
Epoch 14/50
876/876 [==============================] - ETA: 0s - loss: 0.1648 - accuracy: 0.9374 - miou: 0.7885
Epoch 00014: val_miou improved from 0.64721 to 0.64835, saving model to /content/drive/My Drive/Unet.best.
hdf5
876/876 [==============================] - 1045s 1s/step - loss: 0.1648 - accuracy: 0.9374 - miou: 0.7885
- val_loss: 0.3791 - val_accuracy: 0.8825 - val_miou: 0.6484
Epoch 15/50
876/876 [==============================] - ETA: 0s - loss: 0.1608 - accuracy: 0.9390 - miou: 0.7925
Epoch 00015: val_miou did not improve from 0.64835
876/876 [==============================] - 1038s 1s/step - loss: 0.1608 - accuracy: 0.9390 - miou: 0.7925
- val_loss: 0.3887 - val_accuracy: 0.8822 - val_miou: 0.6359
Epoch 16/50
876/876 [==============================] - ETA: 0s - loss: 0.1579 - accuracy: 0.9401 - miou: 0.7944
Epoch 00016: val_miou did not improve from 0.64835
876/876 [==============================] - 1040s 1s/step - loss: 0.1579 - accuracy: 0.9401 - miou: 0.7944
- val_loss: 0.3978 - val_accuracy: 0.8827 - val_miou: 0.6441
Epoch 17/50
876/876 [==============================] - ETA: 0s - loss: 0.1546 - accuracy: 0.9416 - miou: 0.7991
Epoch 00017: val_miou did not improve from 0.64835
876/876 [==============================] - 1032s 1s/step - loss: 0.1546 - accuracy: 0.9416 - miou: 0.7991
- val_loss: 0.4033 - val_accuracy: 0.8802 - val_miou: 0.6464
Epoch 18/50
876/876 [==============================] - ETA: 0s - loss: 0.1485 - accuracy: 0.9441 - miou: 0.8052
Epoch 00018: val_miou did not improve from 0.64835
876/876 [==============================] - 1039s 1s/step - loss: 0.1485 - accuracy: 0.9441 - miou: 0.8052
- val_loss: 0.4112 - val_accuracy: 0.8809 - val_miou: 0.6470
Epoch 19/50
876/876 [==============================] - ETA: 0s - loss: 0.1439 - accuracy: 0.9460 - miou: 0.8114
Epoch 00019: val_miou did not improve from 0.64835
876/876 [==============================] - 1041s 1s/step - loss: 0.1439 - accuracy: 0.9460 - miou: 0.8114
- val_loss: 0.4302 - val_accuracy: 0.8815 - val_miou: 0.6365
Epoch 20/50
876/876 [==============================] - ETA: 0s - loss: 0.1435 - accuracy: 0.9459 - miou: 0.8127
Epoch 00020: val_miou did not improve from 0.64835
876/876 [==============================] - 1035s 1s/step - loss: 0.1435 - accuracy: 0.9459 - miou: 0.8127
- val_loss: 0.4168 - val_accuracy: 0.8793 - val_miou: 0.6423
Epoch 00020: early stopping
--- 20962.57428073883 seconds ---
```

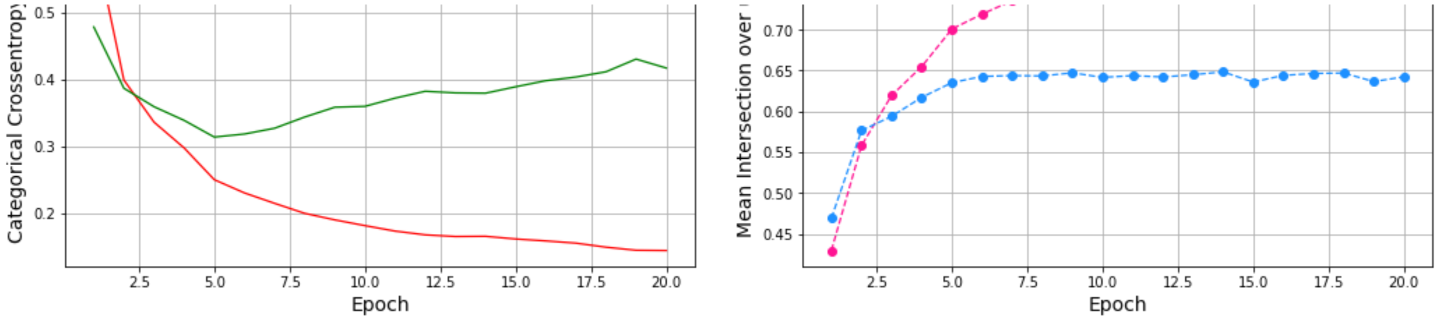
U-Net Training Results

In [ ]:

```
# Training_result
plot_training_result(history)
```







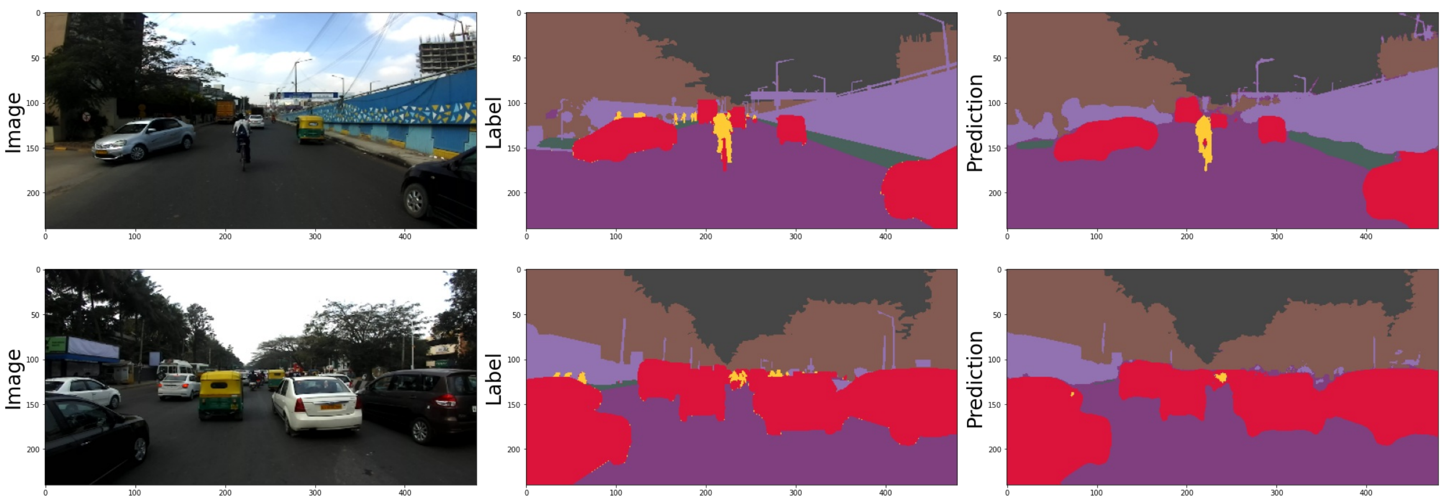
- **The Lowest value of Validation Categorical Crossentopy is 0.3134 which is at epoch-5 as above in the Graph.**
- **The Best Value of Validation Mean Intersection Over Union is 0.6484 which is at epoch-14 as above in the Graph.**
- **Keras callback ModelCheckpoint is used to save the best Model during Training to avoid overfitting.**

U-Net Prediction on Train Data

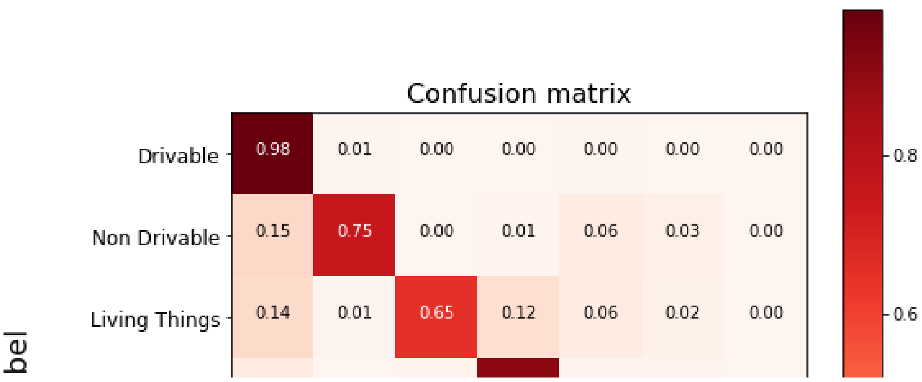
In [ ]:

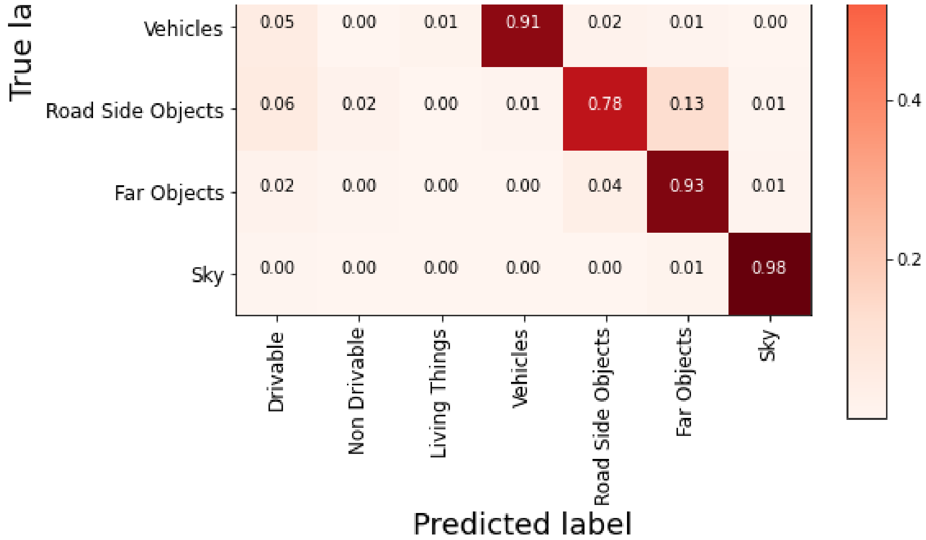
```
# Train Prediction
Miou, Accuracy, cf_matrix = predict_for("Train_data")
```

```
Total number of samples in Train_data : 10016

Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score      |
-----------------------

   MIOU Score: 0.6642

-----------------------
|   Accuracy Score    |
-----------------------

   Accuracy Score: 0.9306

-----------------------
|  Confusion Matrix    |
-----------------------
```



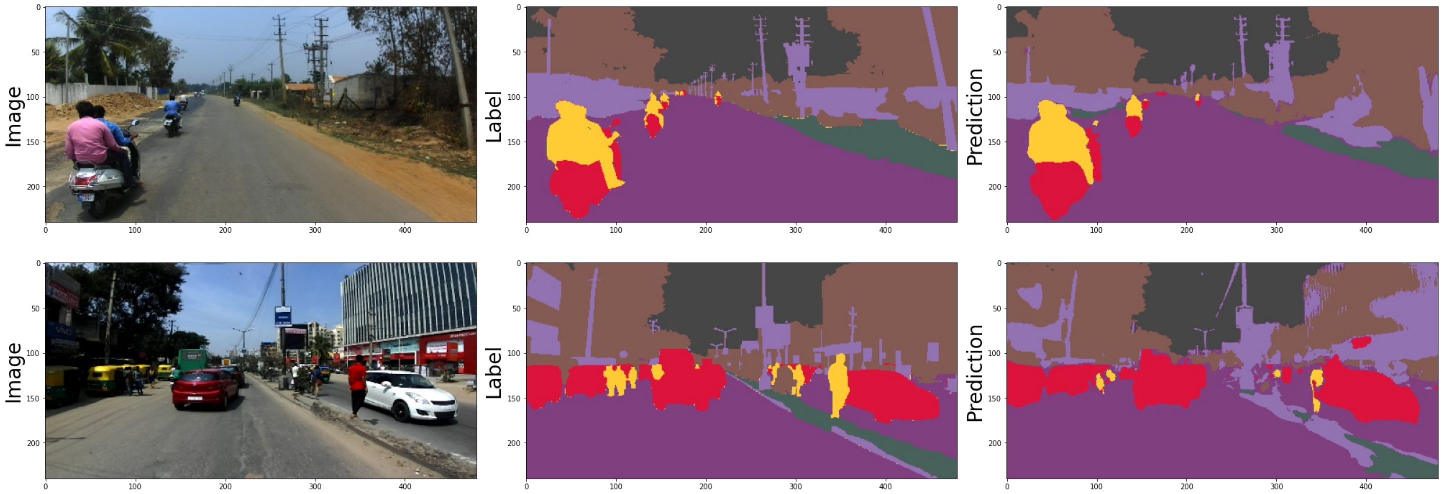


U-Net Prediction on Validation Data

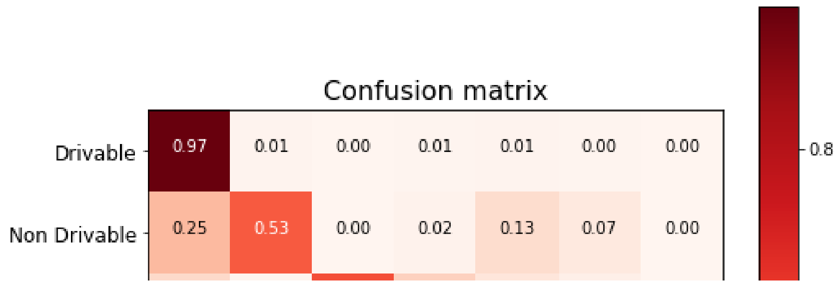
In [ ]:

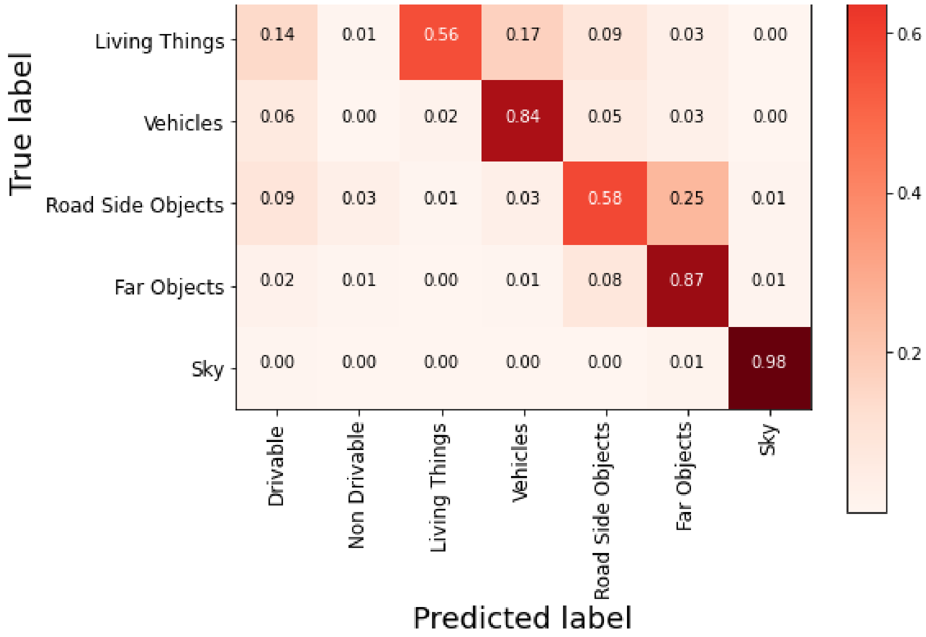
```
# Validation Prediction
Miou, Accuracy, cf_matrix = predict_for("Val_data")
```

```
Total number of samples in Val_data : 2036

Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score      |
-----------------------

   MIOU Score: 0.5843

-----------------------
|    Accuracy Score    |
-----------------------

   Accuracy Score: 0.876

-----------------------
|  Confusion Matrix    |
-----------------------
```



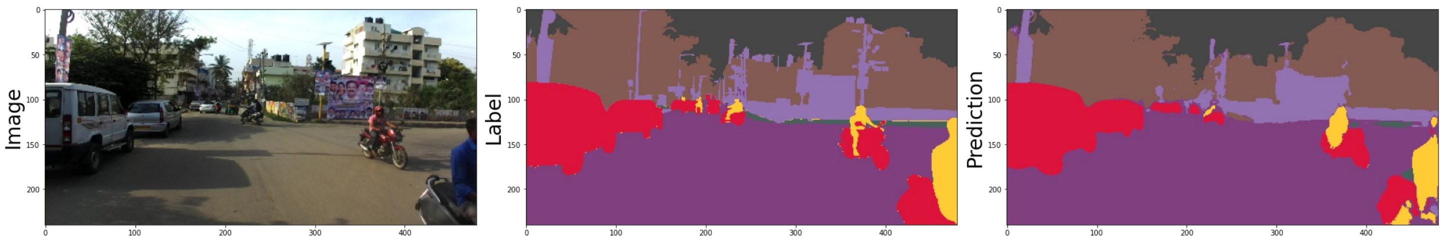


U-Net Prediction on Test Data

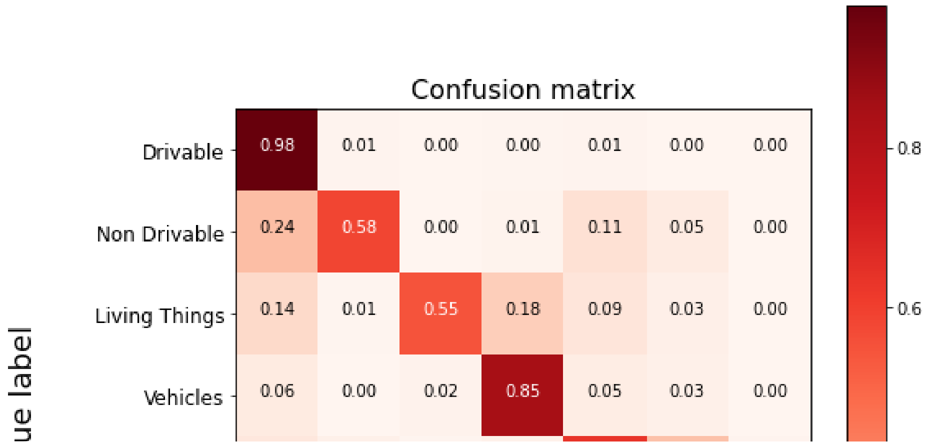
In [ ]:

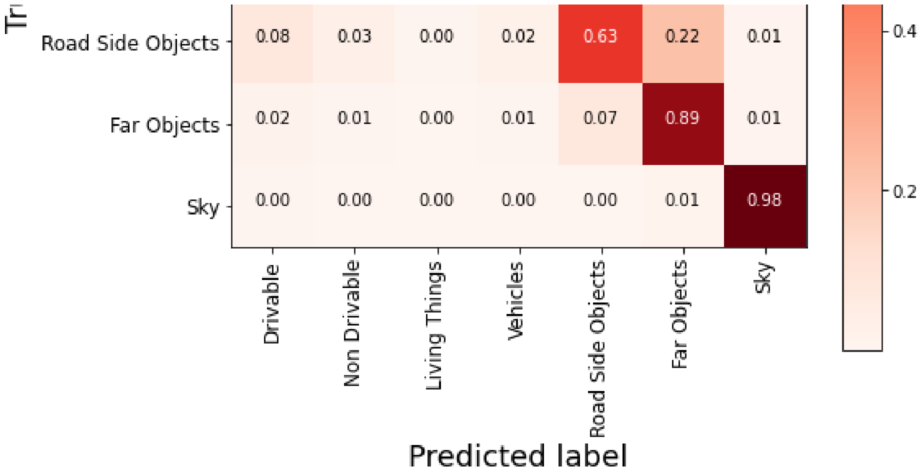
```
# Test Prediction
Miou, Accuracy, cf_matrix = predict_for("Test_data")
```

```
Total number of samples in Test_data : 4011

Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score     |
-----------------------

   MIOU Score: 0.5979

-----------------------
|   Accuracy Score    |
-----------------------

   Accuracy Score: 0.8888

-----------------------
|  Confusion Matrix   |
-----------------------
```





Transfer Learning with U-Net

- **Transfer learning refers to the process of using pre-trained model of one problem to solve other new second problem where the knowledge gained by pre-trained model is applied to solve a different but related problem.**
- **The main advantages of Transfer learning are saving training time, better performance of neural networks and no need for lot of data.**
- **Imagenet Pre-Trained Restnet50 with some alteration on U-Net is used below to solve the problem.**

Importing and Installing segmentation model package

In [ ]:

```
# reference: https://github.com/qubvel/segmentation_models
! pip install tensorflow==2.1.0
! pip install -U segmentation-models
import segmentation_models as sm
import tensorflow.keras
tensorflow.keras.backend.set_image_data_format('channels_last')
```

```
Segmentation Models: using `tf.keras` framework.
```

Training Restnet50 + U-Net with Imagenet Pre-Trained weigths.

In [ ]:

```
start_time = time()
batch_size, epochs = 16, 50
model = sm.Unet('resnet50',classes=7,input_shape=(224, 480,3),activation='softmax')
tensorboard, filepath = TensorBoard(log_dir=root+"logs/unet_img_resnet50_nlrr{}".format(str(time())[:10])
),root+"Unet_imgnet_resnet50_nlrr.hdf5"
steps_per_epoch,validation_steps=int((len(train_img_files1)+len(train_img_files2))/batch_size),int((len(v
al_img_files1)+len(val_img_files2))/batch_size)
model.compile(optimizer = tf.keras.optimizers.Adam(0.0001), loss = 'categorical_crossentropy',metrics = [
'accuracy',miou])
es = EarlyStopping(monitor='val_miou', mode='max', verbose=1, patience=5)
checkpoint = ModelCheckpoint(filepath, monitor='val_miou', verbose=2, save_best_only=True, mode='max')
history_tf=model.fit_generator(train_batch_generator(batch_size,epochs), steps_per_epoch=steps_per_epoch,
epochs=epochs, verbose=1, validation_data=val_batch_generator(batch_size,epochs),
                              validation_steps=validation_steps, callbacks=[checkpoint,es,tensorboard])
print("--- %s seconds ---" % (time() - start_time))
```

```
Train for 876 steps, validate for 127 steps
Epoch 1/50
875/876 [============================>.] - ETA: 0s - loss: 0.5070 - accuracy: 0.8557 - miou: 0.5958
Epoch 00001: val_miou improved from -inf to 0.56564, saving model to /content/drive/My Drive/Unet_imgnet_r
esnet50_nlrr.hdf5
876/876 [==============================] - 552s 631ms/step - loss: 0.5067 - accuracy: 0.8558 - miou: 0.595
9 - val_loss: 0.4781 - val_accuracy: 0.8363 - val_miou: 0.5656
Epoch 2/50
875/876 [============================>.] - ETA: 0s - loss: 0.2468 - accuracy: 0.9109 - miou: 0.7180
Epoch 00002: val_miou improved from 0.56564 to 0.66625, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 544s 621ms/step - loss: 0.2467 - accuracy: 0.9109 - miou: 0.718
0 - val_loss: 0.2910 - val_accuracy: 0.8918 - val_miou: 0.6663
Epoch 3/50
875/876 [============================>.] - ETA: 0s - loss: 0.2024 - accuracy: 0.9255 - miou: 0.7545
```

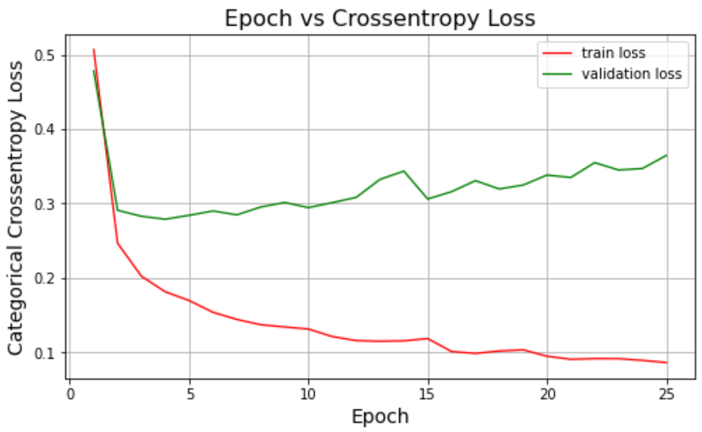
```
Epoch 00003: val_miou improved from 0.66625 to 0.67648, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 555s 634ms/step - loss: 0.2024 - accuracy: 0.9255 - miou: 0.754
6 - val_loss: 0.2830 - val_accuracy: 0.8970 - val_miou: 0.6765
Epoch 4/50
875/876 [============================>.] - ETA: 0s - loss: 0.1815 - accuracy: 0.9326 - miou: 0.7750
Epoch 00004: val_miou improved from 0.67648 to 0.67879, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 551s 629ms/step - loss: 0.1815 - accuracy: 0.9326 - miou: 0.775
0 - val_loss: 0.2790 - val_accuracy: 0.9019 - val_miou: 0.6788
Epoch 5/50
875/876 [============================>.] - ETA: 0s - loss: 0.1698 - accuracy: 0.9366 - miou: 0.7863
Epoch 00005: val_miou improved from 0.67879 to 0.67916, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 550s 628ms/step - loss: 0.1698 - accuracy: 0.9366 - miou: 0.786
3 - val_loss: 0.2843 - val_accuracy: 0.9023 - val_miou: 0.6792
Epoch 6/50
875/876 [============================>.] - ETA: 0s - loss: 0.1540 - accuracy: 0.9424 - miou: 0.8021
Epoch 00006: val_miou improved from 0.67916 to 0.68699, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 552s 630ms/step - loss: 0.1540 - accuracy: 0.9424 - miou: 0.802
1 - val_loss: 0.2901 - val_accuracy: 0.9058 - val_miou: 0.6870
Epoch 7/50
875/876 [============================>.] - ETA: 0s - loss: 0.1444 - accuracy: 0.9459 - miou: 0.8122
Epoch 00007: val_miou improved from 0.68699 to 0.68837, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 555s 634ms/step - loss: 0.1444 - accuracy: 0.9459 - miou: 0.812
3 - val_loss: 0.2849 - val_accuracy: 0.9055 - val_miou: 0.6884
Epoch 8/50
875/876 [============================>.] - ETA: 0s - loss: 0.1373 - accuracy: 0.9485 - miou: 0.8189
Epoch 00008: val_miou improved from 0.68837 to 0.68987, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 548s 626ms/step - loss: 0.1373 - accuracy: 0.9485 - miou: 0.818
9 - val_loss: 0.2954 - val_accuracy: 0.9050 - val_miou: 0.6899
Epoch 9/50
875/876 [============================>.] - ETA: 0s - loss: 0.1343 - accuracy: 0.9495 - miou: 0.8231
Epoch 00009: val_miou did not improve from 0.68987
876/876 [==============================] - 544s 621ms/step - loss: 0.1343 - accuracy: 0.9495 - miou: 0.823
1 - val_loss: 0.3015 - val_accuracy: 0.9044 - val_miou: 0.6880
Epoch 10/50
875/876 [============================>.] - ETA: 0s - loss: 0.1315 - accuracy: 0.9505 - miou: 0.8264
Epoch 00010: val_miou did not improve from 0.68987
876/876 [==============================] - 546s 623ms/step - loss: 0.1315 - accuracy: 0.9505 - miou: 0.826
4 - val_loss: 0.2946 - val_accuracy: 0.9065 - val_miou: 0.6896
Epoch 11/50
875/876 [============================>.] - ETA: 0s - loss: 0.1214 - accuracy: 0.9542 - miou: 0.8363
Epoch 00011: val_miou improved from 0.68987 to 0.69082, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 552s 630ms/step - loss: 0.1214 - accuracy: 0.9542 - miou: 0.836
3 - val_loss: 0.3013 - val_accuracy: 0.9064 - val_miou: 0.6908
Epoch 12/50
875/876 [============================>.] - ETA: 0s - loss: 0.1160 - accuracy: 0.9562 - miou: 0.8424
Epoch 00012: val_miou improved from 0.69082 to 0.69147, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 548s 626ms/step - loss: 0.1160 - accuracy: 0.9562 - miou: 0.842
4 - val_loss: 0.3083 - val_accuracy: 0.9064 - val_miou: 0.6915
Epoch 13/50
875/876 [============================>.] - ETA: 0s - loss: 0.1151 - accuracy: 0.9566 - miou: 0.8431
Epoch 00013: val_miou did not improve from 0.69147
876/876 [==============================] - 548s 626ms/step - loss: 0.1151 - accuracy: 0.9566 - miou: 0.843
1 - val_loss: 0.3325 - val_accuracy: 0.9054 - val_miou: 0.6908
Epoch 14/50
875/876 [============================>.] - ETA: 0s - loss: 0.1156 - accuracy: 0.9563 - miou: 0.8433
Epoch 00014: val_miou did not improve from 0.69147
876/876 [==============================] - 546s 623ms/step - loss: 0.1157 - accuracy: 0.9563 - miou: 0.843
3 - val_loss: 0.3436 - val_accuracy: 0.9016 - val_miou: 0.6835
Epoch 15/50
875/876 [============================>.] - ETA: 0s - loss: 0.1187 - accuracy: 0.9552 - miou: 0.8408
Epoch 00015: val_miou improved from 0.69147 to 0.69592, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 550s 627ms/step - loss: 0.1187 - accuracy: 0.9552 - miou: 0.840
9 - val_loss: 0.3062 - val_accuracy: 0.9095 - val_miou: 0.6959
Epoch 16/50
875/876 [============================>.] - ETA: 0s - loss: 0.1013 - accuracy: 0.9616 - miou: 0.8593
Epoch 00016: val_miou improved from 0.69592 to 0.69878, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 550s 628ms/step - loss: 0.1013 - accuracy: 0.9616 - miou: 0.859
3 - val_loss: 0.3160 - val_accuracy: 0.9098 - val_miou: 0.6988
Epoch 17/50
875/876 [============================>.] - ETA: 0s - loss: 0.0988 - accuracy: 0.9625 - miou: 0.8627
Epoch 00017: val_miou did not improve from 0.69878
876/876 [==============================] - 546s 623ms/step - loss: 0.0988 - accuracy: 0.9625 - miou: 0.862
7 - val_loss: 0.3308 - val_accuracy: 0.9077 - val_miou: 0.6985
```

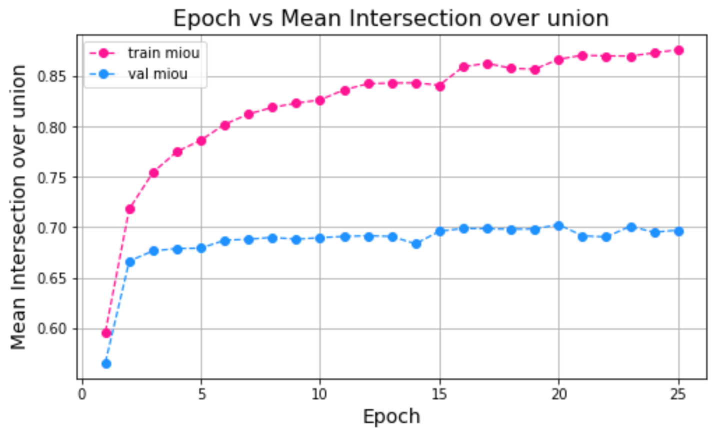
```
Epoch 18/50
875/876 [============================>.] - ETA: 0s - loss: 0.1019 - accuracy: 0.9614 - miou: 0.8577
Epoch 00018: val_miou did not improve from 0.69878
876/876 [==============================] - 545s 622ms/step - loss: 0.1020 - accuracy: 0.9614 - miou: 0.857
7 - val_loss: 0.3196 - val_accuracy: 0.9084 - val_miou: 0.6980
Epoch 19/50
875/876 [============================>.] - ETA: 0s - loss: 0.1036 - accuracy: 0.9607 - miou: 0.8569
Epoch 00019: val_miou did not improve from 0.69878
876/876 [==============================] - 546s 623ms/step - loss: 0.1036 - accuracy: 0.9607 - miou: 0.856
9 - val_loss: 0.3249 - val_accuracy: 0.9086 - val_miou: 0.6982
Epoch 20/50
875/876 [============================>.] - ETA: 0s - loss: 0.0950 - accuracy: 0.9640 - miou: 0.8667
Epoch 00020: val_miou improved from 0.69878 to 0.70262, saving model to /content/drive/My Drive/Unet_imgne
t_resnet50_nlrr.hdf5
876/876 [==============================] - 548s 626ms/step - loss: 0.0950 - accuracy: 0.9640 - miou: 0.866
7 - val_loss: 0.3383 - val_accuracy: 0.9084 - val_miou: 0.7026
Epoch 21/50
875/876 [============================>.] - ETA: 0s - loss: 0.0909 - accuracy: 0.9654 - miou: 0.8707
Epoch 00021: val_miou did not improve from 0.70262
876/876 [==============================] - 545s 622ms/step - loss: 0.0909 - accuracy: 0.9654 - miou: 0.870
7 - val_loss: 0.3351 - val_accuracy: 0.9080 - val_miou: 0.6914
Epoch 22/50
875/876 [============================>.] - ETA: 0s - loss: 0.0918 - accuracy: 0.9650 - miou: 0.8700
Epoch 00022: val_miou did not improve from 0.70262
876/876 [==============================] - 545s 622ms/step - loss: 0.0918 - accuracy: 0.9650 - miou: 0.870
0 - val_loss: 0.3550 - val_accuracy: 0.9082 - val_miou: 0.6903
Epoch 23/50
875/876 [============================>.] - ETA: 0s - loss: 0.0917 - accuracy: 0.9652 - miou: 0.8699
Epoch 00023: val_miou did not improve from 0.70262
876/876 [==============================] - 546s 624ms/step - loss: 0.0917 - accuracy: 0.9652 - miou: 0.869
9 - val_loss: 0.3451 - val_accuracy: 0.9092 - val_miou: 0.7009
Epoch 24/50
875/876 [============================>.] - ETA: 0s - loss: 0.0894 - accuracy: 0.9660 - miou: 0.8729
Epoch 00024: val_miou did not improve from 0.70262
876/876 [==============================] - 547s 625ms/step - loss: 0.0894 - accuracy: 0.9660 - miou: 0.873
0 - val_loss: 0.3471 - val_accuracy: 0.9086 - val_miou: 0.6950
Epoch 25/50
875/876 [============================>.] - ETA: 0s - loss: 0.0865 - accuracy: 0.9671 - miou: 0.8762
Epoch 00025: val_miou did not improve from 0.70262
876/876 [==============================] - 545s 622ms/step - loss: 0.0865 - accuracy: 0.9670 - miou: 0.876
2 - val_loss: 0.3646 - val_accuracy: 0.9076 - val_miou: 0.6971
Epoch 00025: early stopping
--- 13713.721215248108 seconds ---
```

Restnet50 + U-Net Training Results

In [ ]:

```
# Training_result
plot_training_result(history_tf)
```





- **The Lowest value of Validation Categorical Crossentopy is 0.2790 which is at epoch-4 as above in the Graph.**
- **The Best Value of Validation Mean Intersection Over Union is 0.7026 which is at epoch-20 as above in the Graph.**

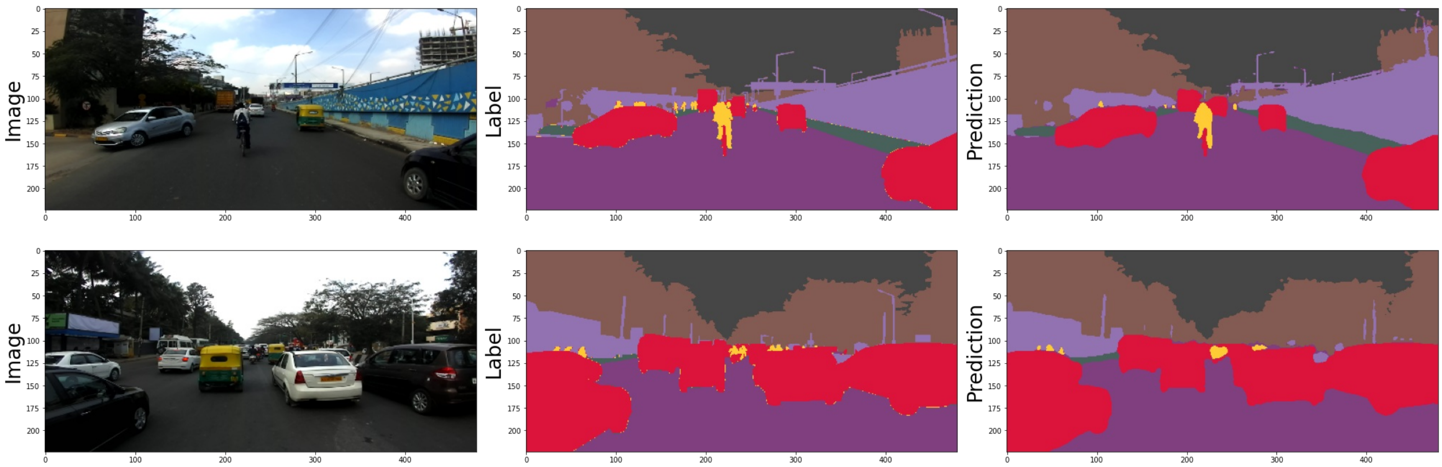
Restnet50 + U-Net Prediction on Train Data

In [ ]:

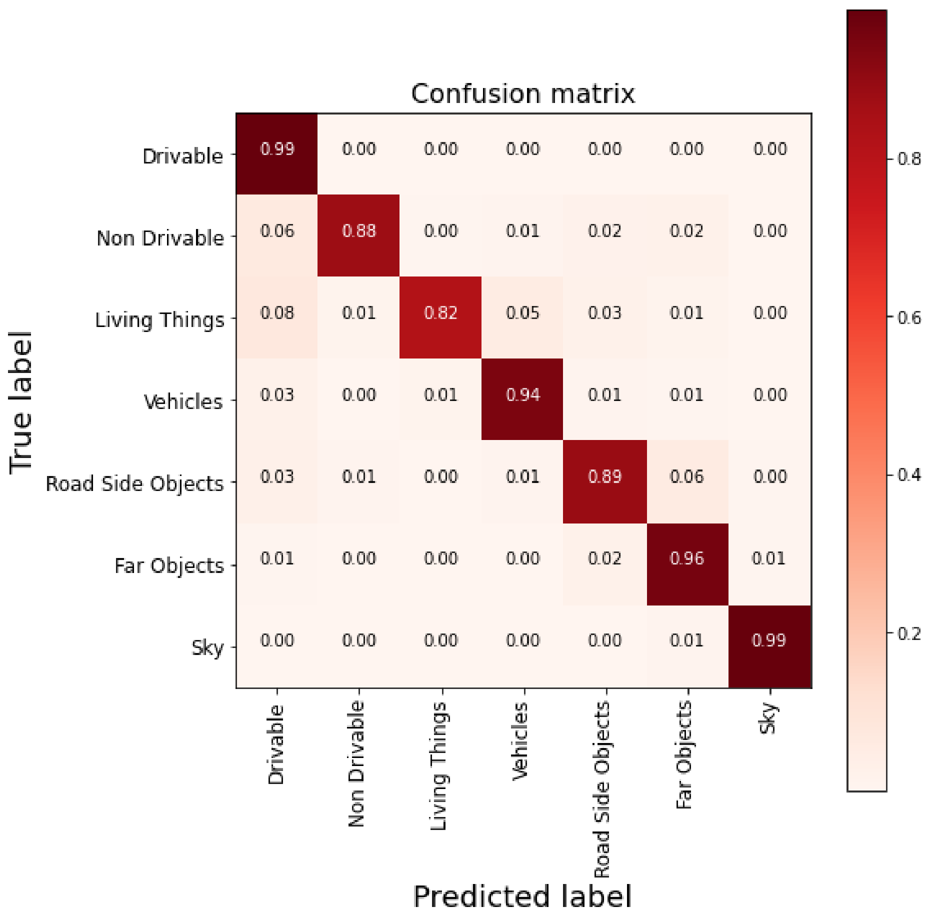
```
Miou, Accuracy, cf_matrix = predict_for("Train_data","/content/drive/My Drive/Unet_imgnet_resnet50_nlrr.h
df5")
```

```
Total number of samples in Train data : 10016
```

```
Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score      |
-----------------------

   MIOU Score: 0.7487

-----------------------
|   Accuracy Score    |
-----------------------

   Accuracy Score: 0.96

-----------------------
|  Confusion Matrix    |
-----------------------
```



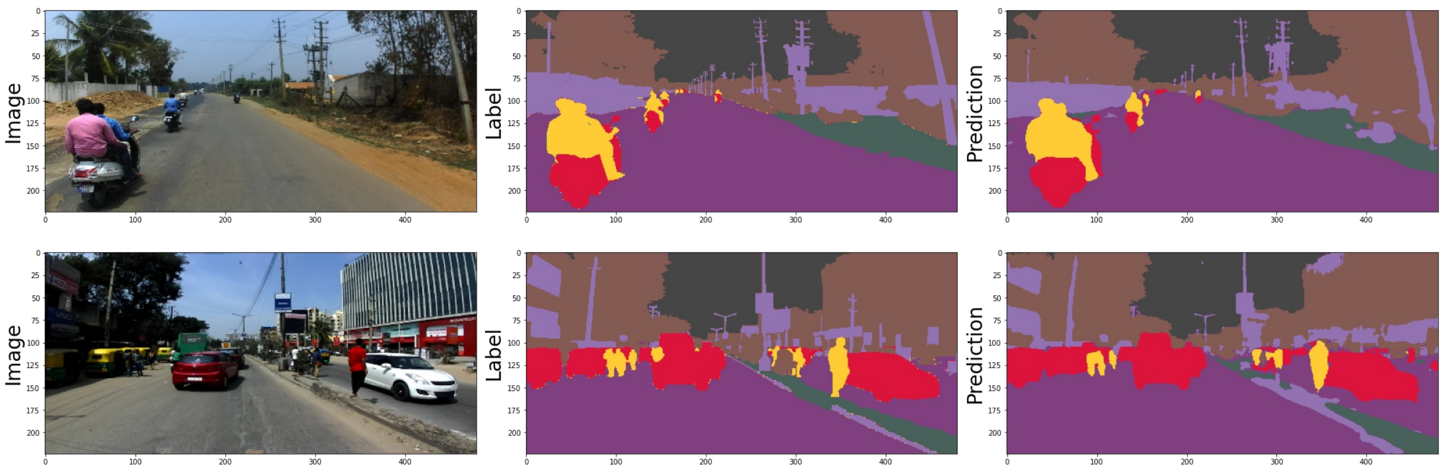
Restnet50 + U-Net Prediction on Validation Data

In [ ]:

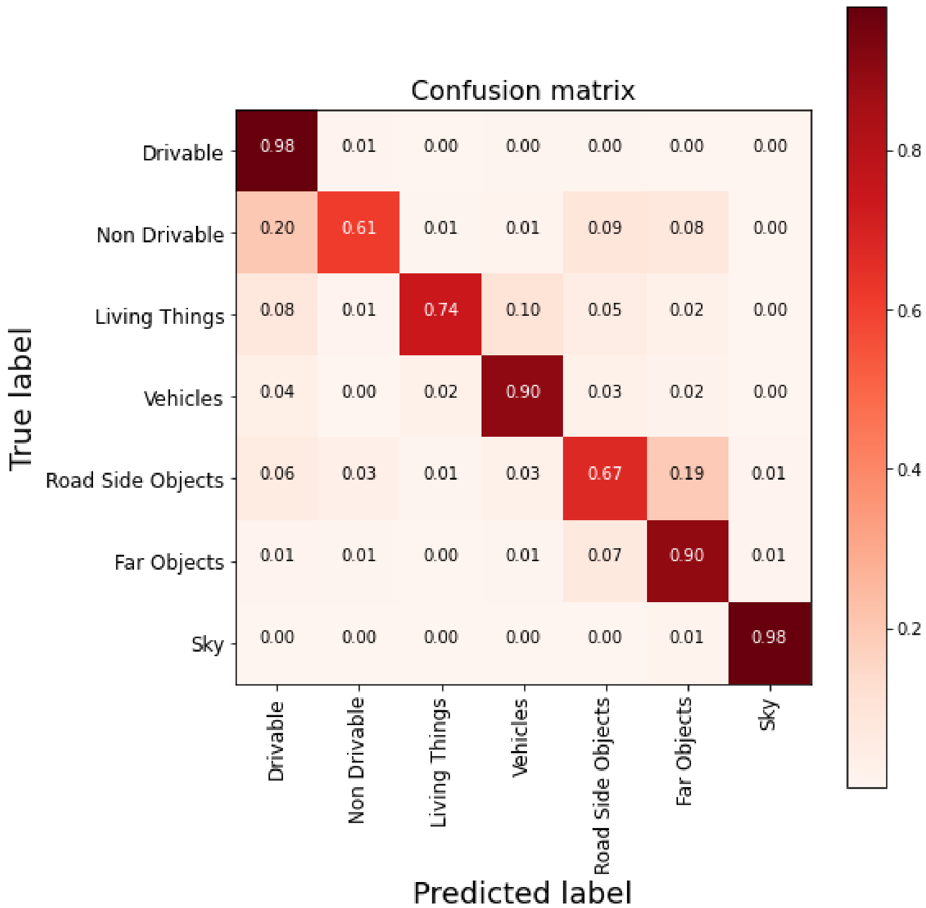
```
Miou, Accuracy, cf_matrix = predict_for("Val_data","/content/drive/My Drive/IID_Files1/New_Model_logs_sav
```

```
e/Unet_imgnet_resnet50_nlrr.hdf5")
```

```
Total number of samples in Test_data : 2036

Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score     |
-----------------------

   MIOU Score: 0.6389

-----------------------
|   Accuracy Score    |
-----------------------

   Accuracy Score: 0.9059

-----------------------
|  Confusion Matrix   |
-----------------------
```

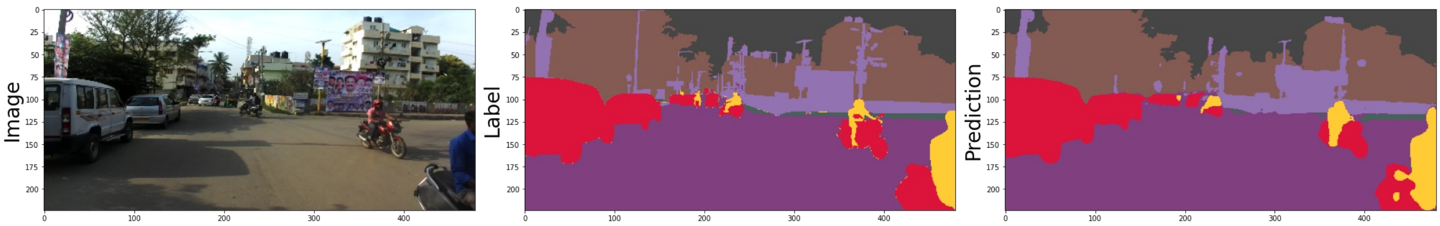


Restnet50 + U-Net Prediction on Test Data

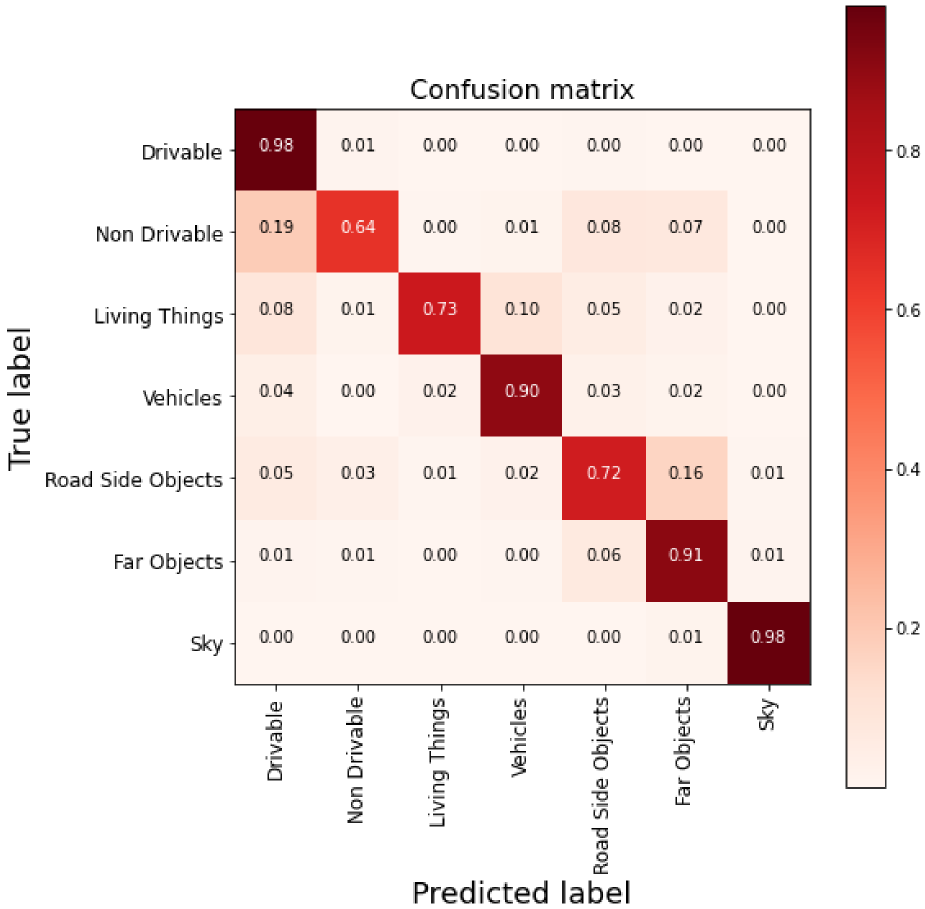
In [ ]:

```
Miou, Accuracy, cf_matrix = predict_for("Test_data","/content/drive/My Drive/IID_Files1/New_Model_logs_sa
ve/Unet_imgnet_resnet50_nlrr.hdf5")
```

```
Total number of samples in Test_data : 4011

Few Segmentation Samples:>>>
```




```
Printing Results:>>

-----------------------
|      MIOU Score      |
-----------------------

   MIOU Score: 0.6496

-----------------------
|   Accuracy Score    |
-----------------------

   Accuracy Score: 0.916

-----------------------
|  Confusion Matrix    |
-----------------------
```



Pretty Tabel

In [ ]:

```
# https://ptable.readthedocs.io/en/latest/tutorial.html
print("\n\t      Performance Table-1")
from prettytable import PrettyTable
T1 = PrettyTable()
T1.field_names = ["U-Net","MIOU", "Accuracy"]
```

```python
T1.add_row(["Train ","0.6642", "0.9306"])
T1.add_row([" ---------- ","----------","----------"])
T1.add_row(["Validation ","0.5843", "0.8760"])
T1.add_row([" ---------- ","----------","----------"])
T1.add_row(["Test ","0.5979", "0.8888"])
print(T1)

print("\n\t      Performance Table-2")
T2 = PrettyTable()
T2.field_names = ["Restnet50 + U-Net ","MIOU", "Accuracy"]
T2.add_row(["Train ","0.7487", "0.9600"])
T2.add_row([" ------------- ","----------","----------"])
T2.add_row(["Validation ","0.6389", "0.9059"])
T2.add_row([" ------------- ","----------","----------"])
T2.add_row(["Test ","0.6496", "0.9160"])
print(T2)
```

```
      Performance Table-1
+--------------+------------+------------+
|    U-Net     |    MIOU    |  Accuracy  |
+--------------+------------+------------+
|    Train     |   0.6642   |   0.9306   |
|  ----------  | ---------- | ---------- |
| Validation   |   0.5843   |   0.8760   |
|  ----------  | ---------- | ---------- |
|    Test      |   0.5979   |   0.8888   |
+--------------+------------+------------+

      Performance Table-2
+--------------------+------------+------------+
| Restnet50 + U-Net  |    MIOU    |  Accuracy  |
+--------------------+------------+------------+
|       Train        |   0.7487   |   0.9600   |
|   -------------    | ---------- | ---------- |
|     Validation     |   0.6389   |   0.9059   |
|   -------------    | ---------- | ---------- |
|       Test         |   0.6496   |   0.9160   |
+--------------------+------------+------------+
```

Conclusion:

- **The U-Net combines the information from the downsampling path and upsampling path to finally obtain general information.**
- **The Deep learning model has misclassified some of the labels between Diving, Non-Driving and Roadside Object, Far Object.**
- **Transfer Learning with some variation on U-Net achieves good performance on Image segmentation when compared to Basic U-net.**
- **More performance can be obtained by training Models with data in high resolution with more powerful hardware resources**