DEEP LEARNING

PROJECT REPORT

MUHAMMAD AFFAN TARIQ

362387

RIME-2021

Part 1: Classification of Satellite Images

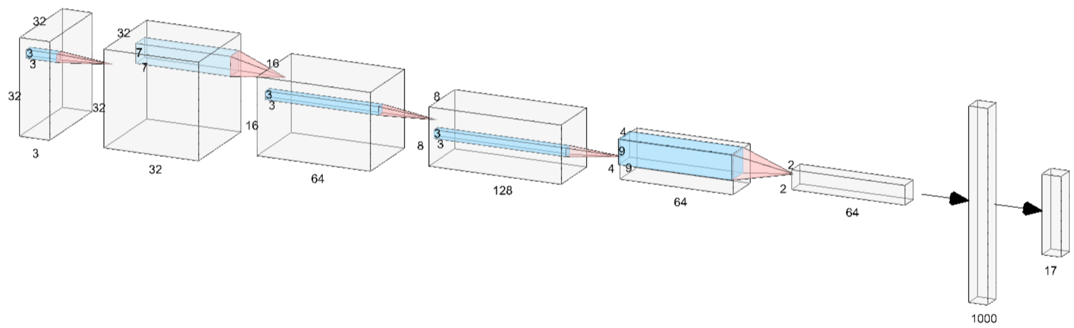


Fig. 1a: CNN Architecture

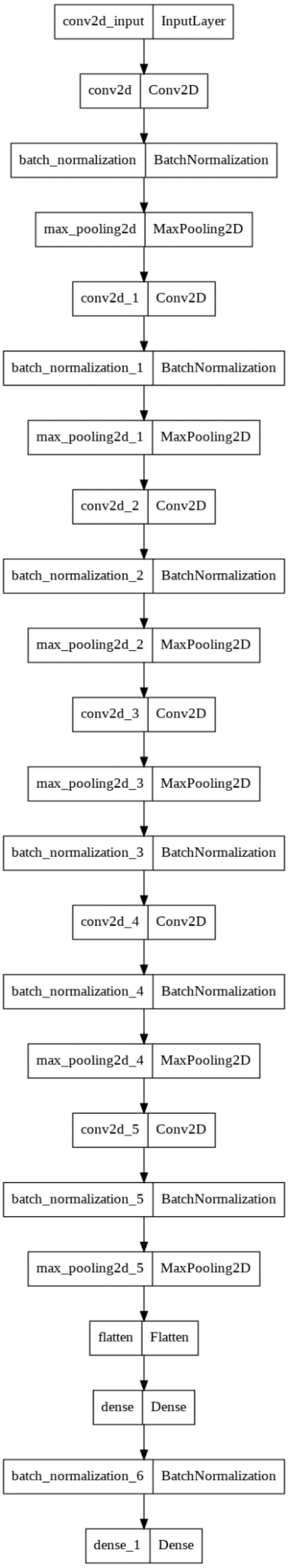


Fig. 1b : CNN Architecture with whole process

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896       
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 32)       128       
 ormalization)                                                   
                                                                 
 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 16, 16, 64)        100416    
                                                                 
 batch_normalization_1 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                 
 batch_normalization_2 (Batc  (None, 8, 8, 128)        512       
 hNormalization)                                                 
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 4, 4, 64)          73792     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 2, 2, 64)         0         
 2D)                                                             
                                                                 
 batch_normalization_3 (Batc  (None, 2, 2, 64)         256       
 hNormalization)                                                 
                                                                 
 conv2d_4 (Conv2D)           (None, 2, 2, 64)          331840    
                                                                 
 batch_normalization_4 (Batc  (None, 2, 2, 64)         256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 1, 1, 64)         0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 1, 1, 64)          200768    
                                                                 
 batch_normalization_5 (Batc  (None, 1, 1, 64)         256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 1, 1, 64)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 64)                0         
                                                                 
 dense (Dense)               (None, 1000)              64000     
```

```
batch_normalization_6 (Batc  (None, 1000)             4000
hNormalization)

dense_1 (Dense)             (None, 17)               17017
```

Fig. 2: CNN Calculations

Dataset:

I have used So2Sat dataset for this part. One of the major obstacles is gaining access to labelled reference data in this project was involving supervised machine learning. This is particularly if remote sensing image analysis is done automatically on a global scale, allowing to address issues on a global scale utilizing cutting-edge machine learning methods, such as urbanization and climate change to satisfy these urgent demands, I have used access to a useful resource, particularly in urban studies.

So2Sat LCZ42 is a benchmark dataset that contains of about 500,000 Sentinel-1 labels for local climatic zones (LCZ) in addition, Sentinel-2 image patches were found in 42 urban agglomerations.10 more minor regions) on the planet. These data were labelled by 15 subject matter specialists after a meticulously planned identifying the process of review and work flow across a six-month period months.

The full dataset consists of 8 Sentinel-1 and 10 Sentinel-2 channels. Alternatively, one can select the rgb subset, which contains only the optical frequency bands of Sentinel-2, rescaled, and encoded as JPEG.

The dataset has a total of 17 different classes shown below which depends upon the type of land present in the images.

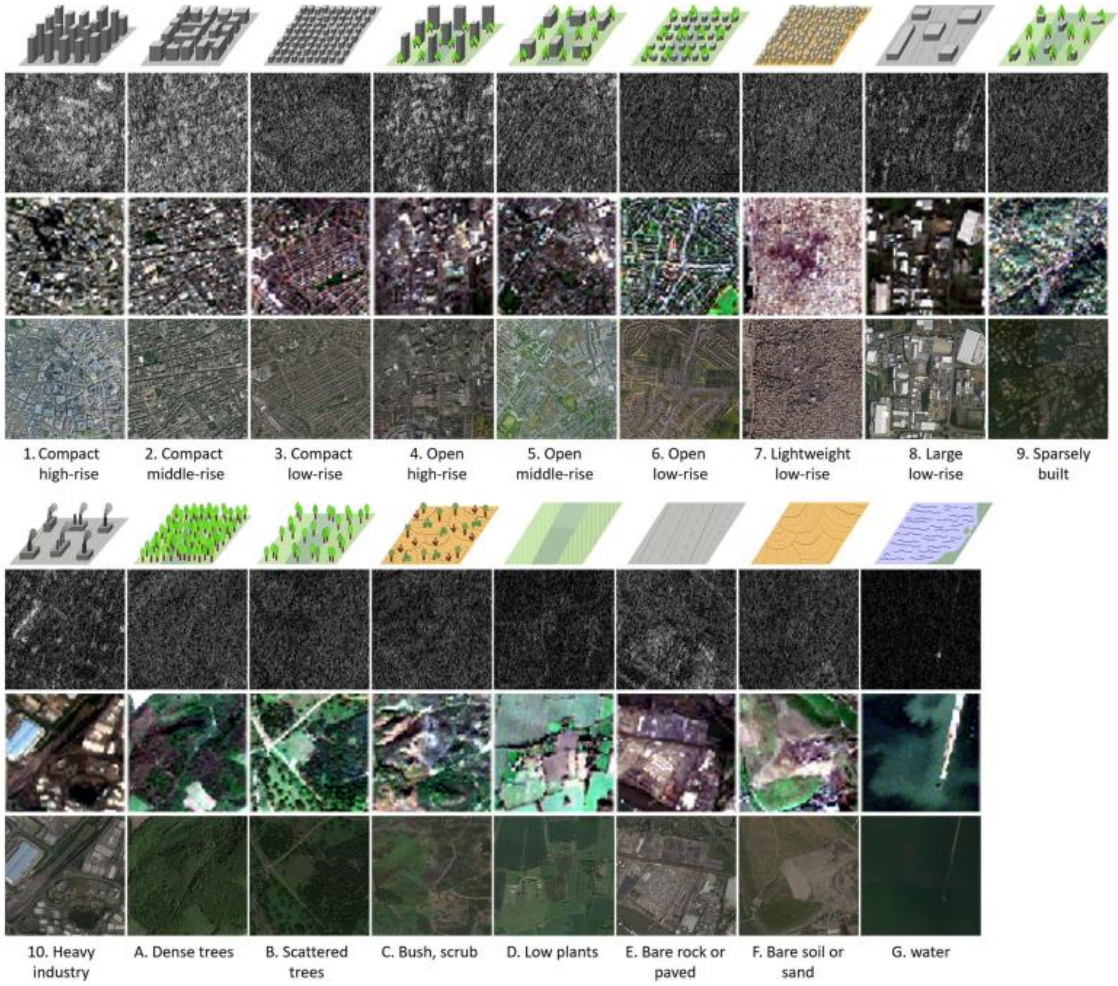


Fig. Ex: Classes in So2Sat Dataset

In each LCZ class, the topmost image is Sentinel-1 scene, the middle one is corresponding Sentinel-2 scene in RGB and lower image is high resolution image from Google Earth just for reference.

The dataset is loaded at first using command:

```
train_set, test_set = tfds.load('so2sat/rgb', split=['train', 'validation'])
```

Labels are assigned and dataset is preprocessed:

```
LABELS = [
    'Compact high-rise', 'Compact mid-rise', 'Compact low-rise',
    'Open high-rise', 'Open mid-rise', 'Open low-rise', 'Lightweight low-rise',
    'Large low-rise', 'Sparsely built', 'Heavy industry', 'Dense trees',
    'Scattered trees', 'Bush or scrub', 'Low plants', 'Bare rock or paved',
```

```
    'Bare soil or sand', 'Water'
]


 X_train= []
 Y_train=[]
# X_train = train_set['image']
# Y_train = train_set['label']
 X_test = []
 Y_test = []
# X_test = test_set['image']
# Y_test = test_set['label']

i=0
j=0
for  exp in train_set:
  image, label = exp['image'], exp['label']
  X_train.append(image)
  # plt.figure(figsize = (20,4))
  # plt.imshow(image)
  Y_train.append(label)
  # plt.show()
  # print('label =', label)
  print ('Train_Image No:', i)
  i += 1

for  exp in test_set:
  image, label = exp["image"], exp["label"]
  X_test.append(image)
  # plt.figure(figsize = (20,4))
  # plt.imshow(image)
  Y_test.append(label)
  # plt.show()
  # print('label =', label)
  print('Test Image No.: ', j)
  j += 1
print(type(X_train))
print(type(Y_train))


for i in range(len(X_train)):
  X_train[i] = X_train[i].numpy()
for i in range(len(X_test)):
  X_test[i] = X_test[i].numpy()
for i in range(len(Y_train)):
  Y_train[i] = Y_train[i].numpy()
for i in range(len(Y_test)):
  Y_test[i] = Y_test[i].numpy()
X_train = np.array(X_train)
```

```
Y_train = np.array(Y_train)
X_test = np.array(X_test)
Y_test = np.array(Y_test)

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
print(X_train)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

print(Y_train.shape)
print(Y_train[0])
# print(Training_Data)
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)
```

Data is normalized by first forming arrays and labels are normalized. The data has been divided into test and train set in the ratio 94:6 to train it on max amount of data. The input to data has shape (32,32,3) for the (r,g,b) channels.

Model and Training:

The following model of CNN is used for its training where my roll number is used as filter sizes : 362387 makes it to 3,7,3,3,9,7 filter sizes consecutively for the 6 conv layers. Max Pool is using (2,2) size filter with stride 1. Padding is done in each layer and relu activation is used. He-uniform is used as kernel weights initializer. After CNN, the extracted features are flattened to form a vector which is passed to ANN for classification which is a 1000 neuron layer. The final layer has 17 neurons corresponding to 17 classes.

The Fig. above show the model images.

```
def cnn():

  cnn = models.Sequential([layers.Conv2D(filters = 32, kernel_size= (3,
3), activation = 'relu', padding ='same',kernel_initializer='he_uniform
',strides=1, input_shape=(32,32,3)),
                         layers.BatchNormalization(),
                         layers.MaxPooling2D((2,2),padding='same'),
                         layers.Conv2D(filters = 64, kernel_size= (7,7)
, kernel_initializer='he_uniform',strides=1, padding ='same', activatio
n = 'relu'),
                         layers.BatchNormalization(),
                         layers.MaxPooling2D((2,2), padding='same'),
                         layers.Conv2D(filters = 128, kernel_size = (3,
 3), activation='relu',strides=1,kernel_initializer='he_uniform',  padd
ing='same'),
                         layers.BatchNormalization(),
                         layers.MaxPooling2D((2,2), padding='same'),
                         layers.Conv2D(filters = 64, kernel_size= (3,3)
,  padding ='same',strides=1,kernel_initializer='he_uniform', activatio
n = 'relu'),
                         layers.MaxPooling2D((2,2),padding='same'),
                         layers.BatchNormalization(),
                         layers.Conv2D(filters = 64, kernel_size= (9,9)
,  padding ='same',strides=1,kernel_initializer='he_uniform', activatio
n = 'relu'),
                         layers.BatchNormalization(),
                         layers.MaxPooling2D((2,2),padding='same'),
                         layers.Conv2D(filters = 64, kernel_size= (7,7)
,  padding ='same',strides=1,kernel_initializer='he_uniform', activatio
n = 'relu'),
                         layers.BatchNormalization(),
                         layers.MaxPooling2D((2,2), padding='same'),
                         layers.Flatten(),
                         layers.Dense(1000, activation = 'relu', use_bi
as =False),
                         layers.BatchNormalization(),
                         layers.Dense(17, activation = 'softmax')
])
  cnn.compile(tf.keras.optimizers.Adam(learning_rate = 0.003),
          loss = 'categorical_crossentropy',
          metrics = ['accuracy'])


  return cnn

cnn= cnn()


from keras.wrappers.scikit_learn import KerasClassifier
```

```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.datasets import make_classification
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnP
lateau
from sklearn.metrics import precision_score, recall_score
losses = []
accuracies = []
precision_list = []
recall_list = []

file_path = '/content/drive/MyDrive/DL Project Part 1 '
Checkpoint_Model = tf.keras.callbacks.ModelCheckpoint(monitor="loss",
                                                      mode = 'min',
                                                      save_best_only =
True,
                                                      verbose = 0,
                                                      filepath=file_pat
h)
early = EarlyStopping(monitor="loss", mode="min", patience=8)
callbacks_list = [Checkpoint_Model, early]
# cnn = cnn()
rounded_labels=np.argmax(Y_test, axis=1)
hist = cnn.fit(X_train, Y_train, validation_split = 0.2,
                         epochs = 2,  verbose = True)


y_pred = np.argmax(cnn.predict(X_test), axis = 1)
precision_list.append(precision_score(rounded_labels, y_pred,
                                           average='micro'))
recall_list.append(recall_score(rounded_labels, y_pred,
                                           average='micro'))

hist = cnn.fit(X_train, Y_train, validation_split = 0.2,
                         epochs = 2,  verbose = True)

y_pred = np.argmax(cnn.predict(X_test), axis = 1)
precision_list.append(precision_score(rounded_labels, y_pred,
                                           average='micro'))
recall_list.append(recall_score(rounded_labels, y_pred,
                                           average='micro'))




hist = cnn.fit(X_train, Y_train, validation_split = 0.2,
                         epochs = 3, callbacks=callbacks_list, verbose
= True)

y_pred = np.argmax(cnn.predict(X_test), axis = 1)
```

```
precision_list.append(precision_score(rounded_labels, y_pred,
                                                average='micro'))
recall_list.append(recall_score(rounded_labels, y_pred,
                                                average='micro'))


cnn.load_weights(file_path)
score = cnn.evaluate(X_test,Y_test, verbose = 1)
Y_pred = cnn.predict(X_test)
losses.append(score[0])
accuracies.append(score[1])
# print('Training Loss and Accuracy')
# import pandas as pd
# pd.# plt.show()
```

As shown in the code above, after making the model, I have imported drive to connect to collab to save weights and model there. I have run it in 7 epochs with precision and recall values calculated after every 2 epochs and after 3 epochs for the last time where the weights are also saved. Fig. 2 show the calculations of parameters and output shape for every step of CNN.

The following code is used to plot Training Loss and Accuracy:

```
%matplotlib inline

train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
epochs = range(len(train_acc))

plt.plot(epochs,train_loss,'r', label='train_loss')
plt.plot(epochs,val_loss,'b', label='val_loss')
plt.title('train_loss vs val_loss')
plt.legend()
plt.figure()

plt.plot(epochs,train_acc,'r', label='train_acc')
plt.plot(epochs,val_acc,'b', label='val_acc')
plt.title('train_acc vs val_acc')
plt.legend()
plt.figure()
```

The graphs are shown in Fig.3.

Weights and model are saved as hdf5 files.

```
cnn.save_weights('cnn_weights.h5')

cnn.save('cnn.h5')
```

Results:

Precision, Recall, Accuracy values, ROC curves and confusion matrix are given from Fig. 4 to 6. The Classification report is as following:

```
classification Report:
              precision    recall  f1-score   support

           0       0.13      0.08      0.10       256
           1       0.38      0.48      0.42      1254
           2       0.46      0.17      0.25      2353
           3       0.38      0.75      0.50       849
           4       0.25      0.22      0.23       757
           5       0.69      0.23      0.34      1906
           6       0.00      0.00      0.00       474
           7       0.72      0.70      0.71      3395
           8       0.65      0.44      0.52      1914
           9       0.13      0.32      0.19       860
          10       0.96      0.55      0.70      2287
          11       0.14      0.18      0.16       382
          12       0.03      0.02      0.03      1202
          13       0.51      0.89      0.65      2747
          14       0.13      0.45      0.21       202
          15       0.21      0.16      0.18       672
          16       0.75      0.98      0.85      2609

    accuracy                           0.51     24119
   macro avg       0.38      0.39      0.35     24119
weighted avg       0.54      0.51      0.49     24119
```

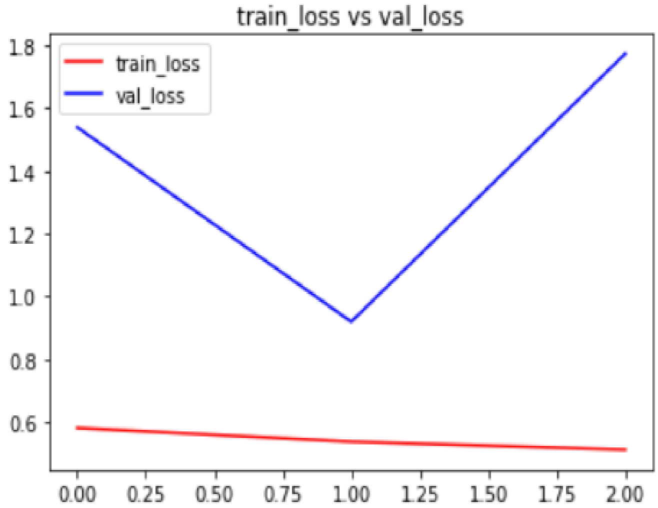
It shows class wise precision, recall, F-score and support values which are then taken macro averages and weighted averages and those values are given at the bottom.

The codes for it are:

```
from sklearn.metrics import confusion_matrix
results = np.argmax(cnn.predict(X_test), axis = 1)
# results = (cnn.predict(X_test) > 0.5).astype("int32")
cm = confusion_matrix(np.where(Y_test==1)[1], results)
cm_df = pd.DataFrame(cm, index = LABELS, columns= LABELS)

final_cm = cm_df
plt.figure(figsize = (10,10))
sns.heatmap(final_cm, annot = True,cmap='Greys',cbar=False,linewidth=2,
fmt='d')
```

```
plt.title('Satellite Data Classification')
plt.ylabel('True class')
plt.xlabel('Prediction class')
plt.show()
from sklearn.metrics import  classification_report
# y_pred = cnn.predict(X_test)

# print(y_pred[:5])
Y_pred_classes = [np.argmax(element) for element in Y_pred]
# cm = confusion_matrix()
rounded_labels=np.argmax(Y_test, axis=1)
print('classification Report: \n', classification_report(rounded_labels
,Y_pred_classes))
classes = 17
Y_pred_ravel = Y_pred.ravel()
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(classes):
  fpr[i], tpr[i],_ = roc_curve(Y_test[:,i], Y_pred[:,i])
  roc_auc[i] = auc(fpr[i], tpr[i])
colors = cycle([
    'red', 'green','black','blue', 'yellow','purple','orange','purple',
 'white', 'magenta',
    'darkred', 'midnightblue', 'navy', 'thistle', 'indigo', 'gold','gre
enyellow'

])

for i, color in zip(range(classes), colors):
  plt.plot(fpr[i],tpr[i], color=color, lw=2, label='ROC curve of class
{0}'''.format(LABELS[i]) )


# f.set_figwidth(4)
# f.set_figheight(4)
plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0, 3.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="best")
plt.figure(figsize= (10,10))
plt.show()
print(precision_list)
```

```
print(recall_list)
```



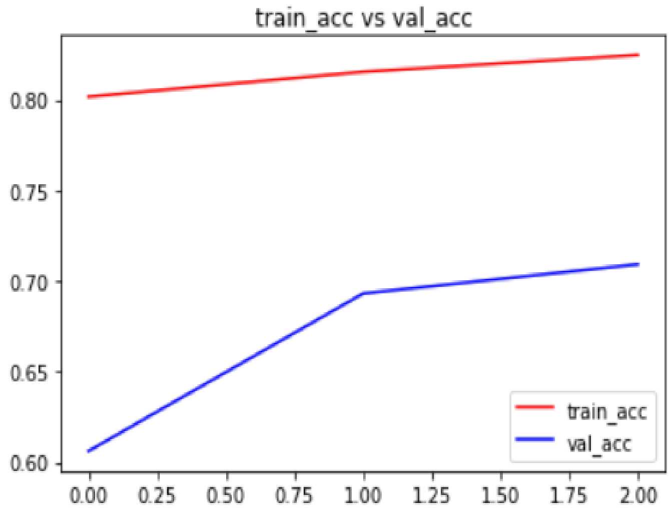


Fig. 3: Training Loss and Accuracy

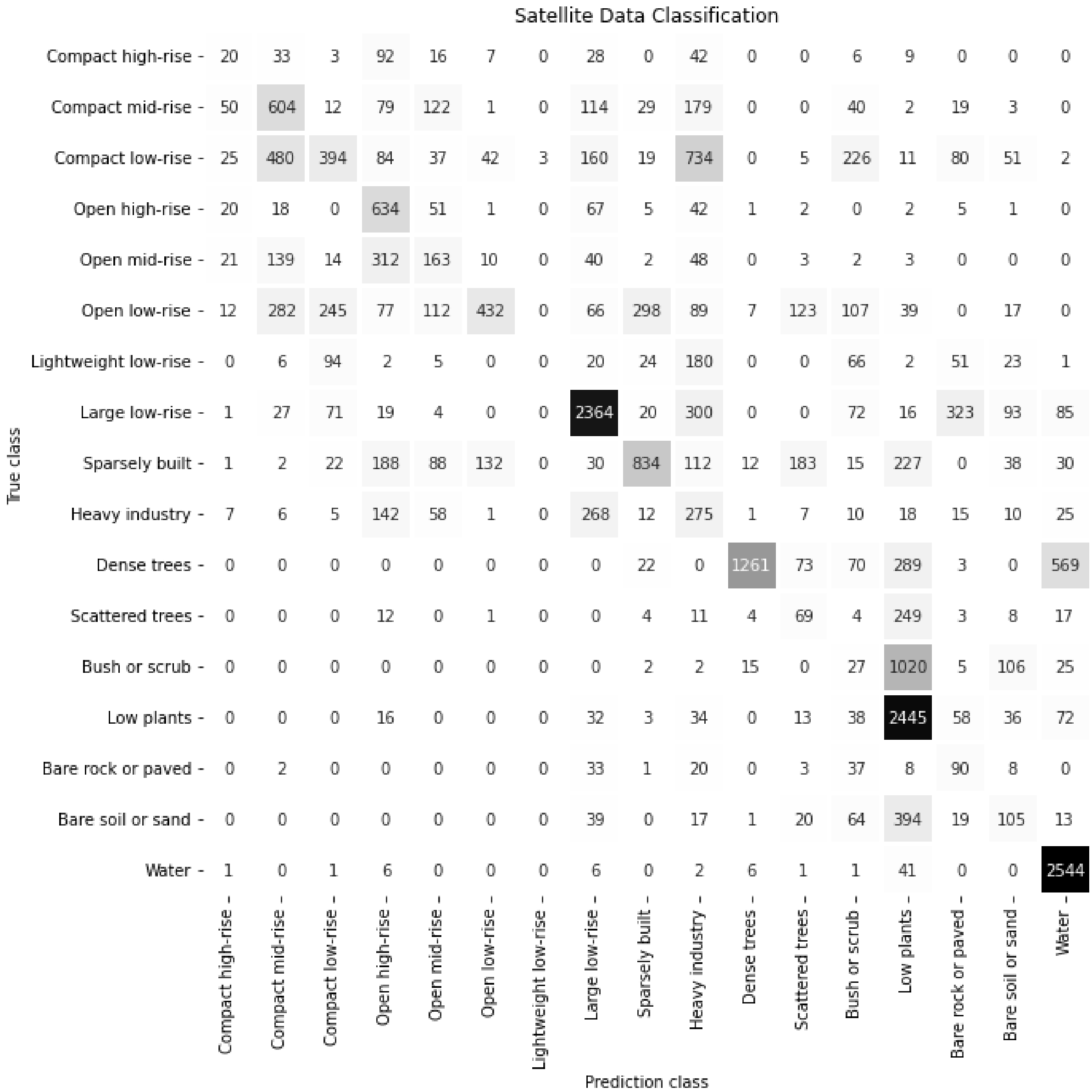


Fig. 4: Confusion Matrix for Satellite Dataset

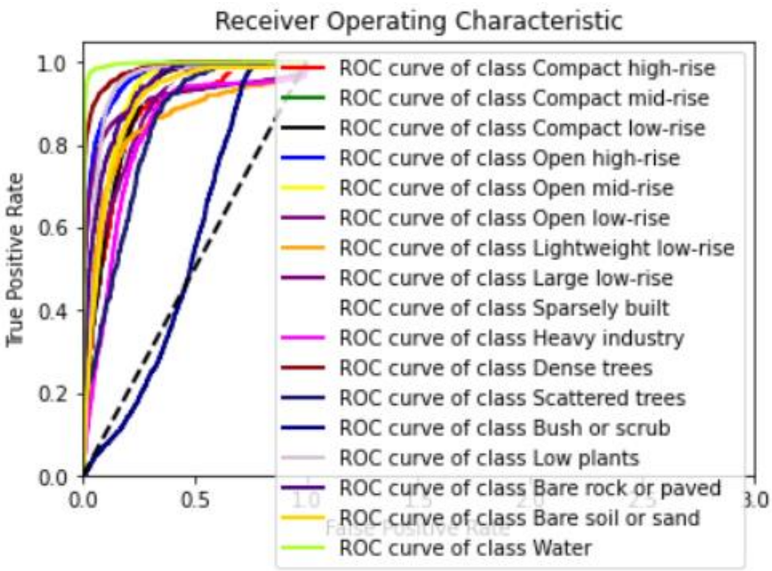


Fig. 5:ROC Curve

```
precision_list
```

```
[0.4184253078485841, 0.45333554459140096, 0.5083544093867906]
```

```
recall_list
```

```
[0.4184253078485841, 0.45333554459140096, 0.5083544093867906]
```

Fig. 6: Precision Recall Curve Results

Part 2: Segmentation and Ensemble of Self Driving Car Dataset

Dataset, Model and Training:

U-Net is a type of CNN designed for quick, precise image segmentation, and I have used it to predict a label for every single pixel in an image - in this case, an image from a self-driving car dataset.

This type of image classification is called semantic image segmentation. It's similar to object detection but where object detection labels objects with bounding boxes that may include pixels that aren't part of the object, semantic image segmentation allows you to predict a precise mask for each object in the image by labeling each pixel in the image with its corresponding class. The word “semantic” here refers to what's being shown, so for example the “Car” class is indicated below by the dark blue mask, and "Person" is indicated with a red mask.

Region-specific labeling is a pretty crucial consideration for self-driving cars, which require a pixel-perfect understanding of their environment so they can change lanes and avoid other cars, or any number of traffic obstacles that can put peoples' lives in danger.

Self-Driving Car dataset is used where I had made separate folders for both training and test sets.

Images are first fed through several convolutional layers which reduce height and width, while growing the number of channels as per roll number sizes.

The contracting path follows a regular CNN architecture, with convolutional layers, their activations, and pooling layers to down sample the image and extract its features. In detail, it consists of the repeated application of two unpadded convolutions, each followed by a rectified linear unit (ReLU) and a 2 x 2 max pooling operation with stride 2 for down sampling. At each down sampling step, the number of feature channels is doubled.

Crop Function crops the image from the contracting path and concatenates it to the current image on the expanding path to create a skip connection.

The expanding path performs the opposite operation of the contracting path, growing the image back to its original size, while shrinking the channels gradually.

In detail, each step in the expanding path upsamples the feature map, followed by a convolution (the transposed convolution). This transposed convolution halves the number of feature channels, while growing the height and width of the image.

Next is a concatenation with the correspondingly cropped feature map from the contracting path, and two convolutions, each followed by a ReLU. You need to perform cropping to handle the loss of border pixels in every convolution.

In the final layer, a 1x1 convolution is used to map each 32-component feature vector to the desired number of classes. The channel dimensions from the previous layer correspond to the number of filters used, so when you use 1x1 convolutions, you can transform that dimension by choosing an appropriate number of 1x1 filters. When this idea is applied to the last layer, you can reduce the channel dimensions to have one layer per class.

The encoder is a stack of various conv_blocks:

Each conv_block() is composed of 2 Conv2D layers with ReLU activations. We will apply Dropout, and MaxPooling2D to some conv_blocks, as you will verify in the following sections, specifically to the last two blocks of the downsampling.

The function will return two tensors:

next_layer: That will go into the next block.

skip_connection: That will go into the corresponding decoding block.

If max_pooling=True, the next_layer will be the output of the MaxPooling2D layer, but the skip_connection will be the output of the previously applied layer(Conv2D or Dropout, depending on the case). Else, both results will be identical.

The decoder, or upsampling block, upsamples the features back to the original image size. At each upsampling level, you'll take the output of the corresponding encoder block and concatenate it before feeding to the next decoder block.

There are two new components in the decoder: up and merge. These are the transpose convolution and the skip connections. In addition, there are two more convolutional layers set to the same parameters as in the encoder.

Here you'll encounter the Conv2DTranspose layer, which performs the inverse of the Conv2D layer.

In semantic segmentation, you need as many masks as you have object classes. In the dataset you're using, each pixel in every mask has been assigned a single integer probability that it belongs to a certain class, from 0 to num_classes-1. The correct class is the layer with the higher probability.

This is different from categorical crossentropy, where the labels should be one-hot encoded (just 0s and 1s). Here, you'll use sparse categorical crossentropy as your loss function, to perform pixel-wise multiclass prediction. Sparse categorical crossentropy is more efficient than other loss functions when you're dealing with lots of classes.

The codes upto training and plots are given below and the model architecture is shown in Fig. 7a and 7b.

```
Training_Path = '/content/drive/MyDrive/Training Set for Self Driving Car/Train-20221224T150606Z-001/Train'
Testing_Path = '/content/drive/MyDrive/Test Set for Self Driving Car/Test-20221224T150605Z-001/Test'
Training_Images = sorted(glob.glob(os.path.join(Training_Path,'images','*.png')))
Training_Masks = sorted(glob.glob(os.path.join(Training_Path,'masks','*.png')))
Testing_Images= sorted(glob.glob(os.path.join(Testing_Path,'images','*.png')))
Testing_Masks = sorted(glob.glob(os.path.join(Testing_Path,'masks','*.png')))
imgs = []
masks = []

for i in range(len(Training_Images)):
  Img = tf.io.read_file(os.path.join(Training_Path, "images", Training_Images[i]))
  Img = tf.image.decode_png(Img, channels = 3)
  Img = tf.image.convert_image_dtype(Img, tf.float32)
  Mask = tf.io.read_file(os.path.join(Training_Path, "masks", Training_Masks[i]))
  Mask = tf.image.decode_png(Mask, channels=3)
  Mask = tf.math.reduce_max(Mask, -1, keepdims= True)
  height, width = Img.shape[0], Img.shape[1]
  h = 96
  w = 128
  Img = tf.image.resize(Img, (h,w), method='nearest')
  Mask = tf.image.resize(Mask, (h,w), method='nearest')
  imgs.append(Img)
  masks.append(Mask)
Train_Images = tf.stack(imgs,axis=0)
Train_Masks = tf.stack(masks,axis=0)
imgs = []
masks = []


for i in range(len(Testing_Images)):
  Img = tf.io.read_file(os.path.join(Testing_Path, "images", Testing_Images[i]))
  Img = tf.image.decode_png(Img, channels = 3)
  Img = tf.image.convert_image_dtype(Img, tf.float32)
```

```
  Mask = tf.io.read_file(os.path.join(Testing_Path, "masks", Testing_Ma
sks[i]))
  Mask = tf.image.decode_png(Mask, channels=3)
  Mask = tf.math.reduce_max(Mask, -1, keepdims= True)
  height, width = Img.shape[0], Img.shape[1]
  h = 96
  w = 128
  Img = tf.image.resize(Img, (h,w), method='nearest')
  Mask = tf.image.resize(Mask, (h,w), method='nearest')
  imgs.append(Img)
  masks.append(Mask)
Test_Images = tf.stack(imgs,axis=0)
Test_Masks = tf.stack(masks,axis=0)


from tensorflow.python.keras import regularizers
from keras.layers import UpSampling2D, Dropout, BatchNormalization
def conv_block(inputs, num_filters, filter_size, dropout_p=0, max_pool
= True):
    """
    Convolutional downsampling block

    Arguments:
        inputs -- Input tensor
        num_filters -- Number of filters for the convolutional layers
        dropout_p -- Dropout probability
        max_pool -
- Use MaxPooling2D to reduce the spatial dimensions of the output volum
e
        filter_size -- Denotes the size of each kernel of filter
    Returns:
        next_layer, skip_connection -
-  Next layer and skip connection outputs
    """
    cnv = Conv2D(num_filters, (filter_size,filter_size), activation = '
relu', padding = 'same', kernel_initializer="he_normal" )(inputs)
    # if dropout_prob > 0 add a dropout layer, with the variable dropou
t_prob as parameter
    cnv = BatchNormalization()(cnv)
    conv = Conv2D(num_filters, # Number of filters
                  (filter_size,filter_size),   # Kernel size
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(cnv)
    conv = BatchNormalization()(conv)
    if dropout_p > 0:
        dropout = Dropout(dropout_p)(conv)
    else:
        dropout = conv
```

```python
    # if max_pooling is True add a MaxPooling2D with 2x2 pool_size
    if max_pool:
        maxpool = MaxPooling2D((2, 2), strides=2)(dropout)
    else:
        maxpool = dropout
    next_layer = maxpool
    skip_connection = dropout

    return next_layer, skip_connection


def upsampling_block(expansive_input, contractive_input, num_filters, f
ilter_size):
    """
    Convolutional upsampling block

    Arguments:
        expansive_input -- Input tensor from previous layer
        contractive_input -- Input tensor from previous skip layer
        num_filters -- Number of filters for the convolutional layers
        filter_size -
- Size of each kernel of filter over a conv2d transpose layer
    Returns:
        conv -- Tensor output
    """

    upsampled_input = Conv2DTranspose(
                 num_filters,
                 (filter_size, filter_size),
                 strides=2,
                 padding="same")(expansive_input)

    # Merge the previous output and the contractive_input
    merge = concatenate([upsampled_input, contractive_input], axis=3)
    conv2d = Conv2D(num_filters,
                  (filter_size, filter_size),
                  activation="relu",
                  padding="same",
                  kernel_initializer="he_normal")(merge)
    conv2d = BatchNormalization()(conv2d)
    conv2d = Conv2D(num_filters,
                  (filter_size, filter_size),
                  activation="relu",
                  padding="same",
                  kernel_initializer="he_normal")(conv2d)

    return conv2d
```

```
def unet(input_size, num_filters, num_classes):
  inputs = Input(input_size)
  conv_block_1 = conv_block(inputs =inputs, num_filters = num_filters*1
, filter_size = 3, dropout_p=0)
  conv_block_2 = conv_block(inputs = conv_block_1[0],num_filters = num_
filters*2, filter_size = 7, dropout_p=0)
  conv_block_3 = conv_block(inputs = conv_block_2[0],num_filters = num_
filters*4, filter_size = 3, dropout_p=0)
  conv_block_4 = conv_block(inputs = conv_block_3[0],num_filters = num_
filters*8, filter_size = 3, dropout_p=0)
  conv_block_5 = conv_block(inputs = conv_block_4[0],num_filters = num_
filters*16, filter_size = 9, dropout_p=0.3)
  # Include a dropout_prob of 0.3 for this layer, and avoid the max_poo
ling layer
  conv_block_6 = conv_block(inputs = conv_block_5[0],num_filters = num_
filters*32, filter_size = 7, dropout_p=0.3, max_pool = False)
  deconv_block_7 = upsampling_block(conv_block_6[0], conv_block_5[1], n
um_filters * 16, filter_size = 3)
  deconv_block_8 = upsampling_block(deconv_block_7, conv_block_4[1], nu
m_filters * 8, filter_size = 7)
  deconv_block_9 = upsampling_block(deconv_block_8, conv_block_3[1], nu
m_filters * 4, filter_size = 3)
  deconv_block_10 = upsampling_block(deconv_block_9, conv_block_2[1], n
um_filters * 2, filter_size = 3)
  deconv_block_11 = upsampling_block(deconv_block_10, conv_block_1[1],
num_filters * 1, filter_size = 9)
  conv2d = Conv2D(num_filters,(7, 7),activation="relu", padding="same")
(deconv_block_11)
  # Add a Conv2D layer with num_classes filter, kernel size of 1 and a
'same' padding
  outputs = Conv2D(num_classes, 1, padding="same")(conv2d)

  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

I have again used my roll number as CNN for 3,7,3,3,9,7 filer sizes for both 6 conv blocks and 6 corresponding deconv blocks.

```
unet= unet(input_size=Train_Images[0].shape, num_filters= 32, num_class
es=12)
```

```
unet.compile(
    tf.keras.optimizers.Adam(learning_rate =0.003),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True
),
    metrics=["accuracy"]
)
path = '/content/drive/MyDrive/DL Project Part 2'
Checkpoint_Model = tf.keras.callbacks.ModelCheckpoint(monitor="loss",
                                                      mode = 'min',
                                                      save_weights_only
 = True,
                                                      save_best_only =
True,
                                                      verbose = 0,
                                                      filepath=path)

EPOCHS = 5

model_history = unet.fit(
    Train_Images,
    Train_Masks,
    epochs=EPOCHS,
    batch_size = 8,
    callbacks = [Checkpoint_Model],
    verbose=True
)
import pandas as pd
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.show()
```

Fig. 8a shows all Unet calculations including parameters and output shapes at each step and Fig. 8b shows the loss and accuracy curves for training.

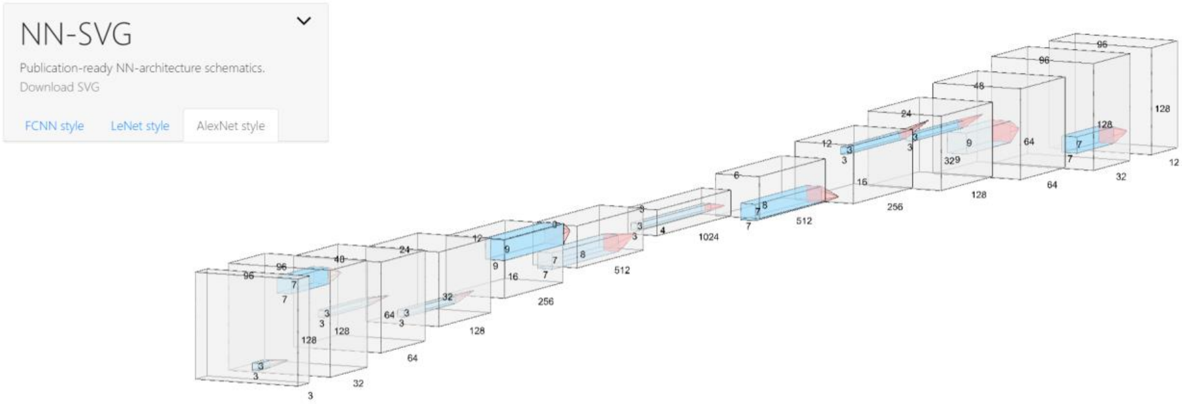


Fig. 7a : Unet Architecture

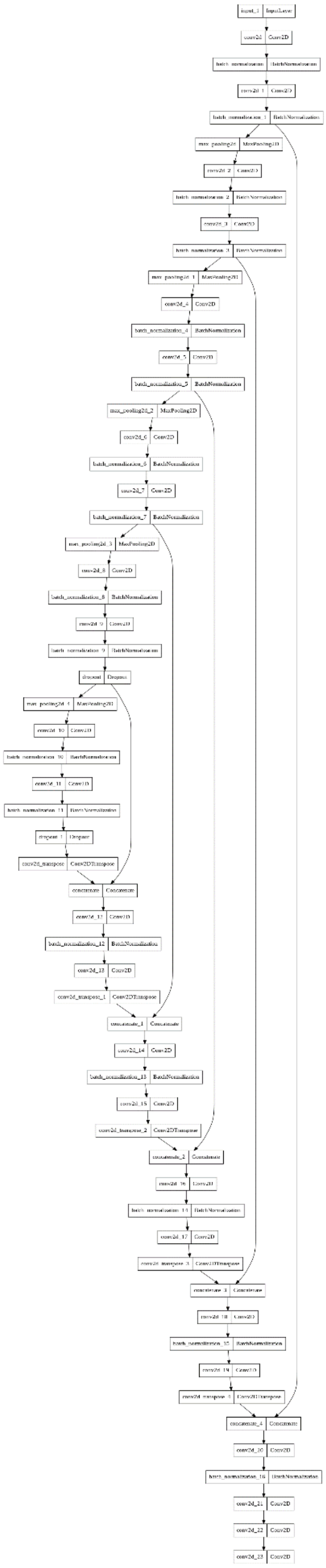


Fig. 7b : Unet Architecture along with process

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 96, 128, 3)  0           []
                                ]

 conv2d (Conv2D)                (None, 96, 128, 32)  896         ['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 96, 128, 32)  128        ['conv2d[0][0]']
 alization)

 conv2d_1 (Conv2D)              (None, 96, 128, 32)  9248        ['batch_normalization[0][0]']

 batch_normalization_1 (BatchNo  (None, 96, 128, 32)  128        ['conv2d_1[0][0]']
 rmalization)

 max_pooling2d (MaxPooling2D)   (None, 48, 64, 32)   0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 48, 64, 64)   100416      ['max_pooling2d[0][0]']

 batch_normalization_2 (BatchNo  (None, 48, 64, 64)  256         ['conv2d_2[0][0]']
 rmalization)

 conv2d_3 (Conv2D)              (None, 48, 64, 64)   200768      ['batch_normalization_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 48, 64, 64)  256         ['conv2d_3[0][0]']
 rmalization)

 max_pooling2d_1 (MaxPooling2D)  (None, 24, 32, 64)  0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 24, 32, 128)  73856       ['max_pooling2d_1[0][0]']

 batch_normalization_4 (BatchNo  (None, 24, 32, 128)  512        ['conv2d_4[0][0]']
 rmalization)

 conv2d_5 (Conv2D)              (None, 24, 32, 128)  147584      ['batch_normalization_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 24, 32, 128)  512        ['conv2d_5[0][0]']
 rmalization)
```

```
 max_pooling2d_2 (MaxPooling2D)  (None, 12, 16, 128)  0
['batch_normalization_5[0][0]']

 conv2d_6 (Conv2D)              (None, 12, 16, 256)  295168
['max_pooling2d_2[0][0]']

 batch_normalization_6 (BatchNo  (None, 12, 16, 256)  1024
['conv2d_6[0][0]']
 rmalization)

 conv2d_7 (Conv2D)              (None, 12, 16, 256)  590080
['batch_normalization_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 12, 16, 256)  1024
['conv2d_7[0][0]']
 rmalization)

 max_pooling2d_3 (MaxPooling2D)  (None, 6, 8, 256)   0
['batch_normalization_7[0][0]']

 conv2d_8 (Conv2D)              (None, 6, 8, 512)    10617344
['max_pooling2d_3[0][0]']

 batch_normalization_8 (BatchNo  (None, 6, 8, 512)   2048
['conv2d_8[0][0]']
 rmalization)

 conv2d_9 (Conv2D)              (None, 6, 8, 512)    21234176
['batch_normalization_8[0][0]']

 batch_normalization_9 (BatchNo  (None, 6, 8, 512)   2048
['conv2d_9[0][0]']
 rmalization)

 dropout (Dropout)              (None, 6, 8, 512)    0
['batch_normalization_9[0][0]']

 max_pooling2d_4 (MaxPooling2D)  (None, 3, 4, 512)   0
['dropout[0][0]']

 conv2d_10 (Conv2D)             (None, 3, 4, 1024)   25691136
['max_pooling2d_4[0][0]']

 batch_normalization_10 (BatchN  (None, 3, 4, 1024)  4096
['conv2d_10[0][0]']
 ormalization)

 conv2d_11 (Conv2D)             (None, 3, 4, 1024)   51381248
['batch_normalization_10[0][0]']

 batch_normalization_11 (BatchN  (None, 3, 4, 1024)  4096
['conv2d_11[0][0]']
 ormalization)

 dropout_1 (Dropout)            (None, 3, 4, 1024)   0
['batch_normalization_11[0][0]']
```

```
 conv2d_transpose (Conv2DTransp  (None, 6, 8, 512)   4719104
['dropout_1[0][0]']
 ose)

 concatenate (Concatenate)      (None, 6, 8, 1024)   0
['conv2d_transpose[0][0]',

'dropout[0][0]']

 conv2d_12 (Conv2D)             (None, 6, 8, 512)    4719104
['concatenate[0][0]']

 batch_normalization_12 (BatchN  (None, 6, 8, 512)   2048
['conv2d_12[0][0]']
 ormalization)

 conv2d_13 (Conv2D)             (None, 6, 8, 512)    2359808
['batch_normalization_12[0][0]']

 conv2d_transpose_1 (Conv2DTran  (None, 12, 16, 256)  6422784
['conv2d_13[0][0]']
 spose)

 concatenate_1 (Concatenate)    (None, 12, 16, 512)  0
['conv2d_transpose_1[0][0]',

'batch_normalization_7[0][0]']

 conv2d_14 (Conv2D)             (None, 12, 16, 256)  6422784
['concatenate_1[0][0]']

 batch_normalization_13 (BatchN  (None, 12, 16, 256)  1024
['conv2d_14[0][0]']
 ormalization)

 conv2d_15 (Conv2D)             (None, 12, 16, 256)  3211520
['batch_normalization_13[0][0]']

 conv2d_transpose_2 (Conv2DTran  (None, 24, 32, 128)  295040
['conv2d_15[0][0]']
 spose)

 concatenate_2 (Concatenate)    (None, 24, 32, 256)  0
['conv2d_transpose_2[0][0]',

'batch_normalization_5[0][0]']

 conv2d_16 (Conv2D)             (None, 24, 32, 128)  295040
['concatenate_2[0][0]']

 batch_normalization_14 (BatchN  (None, 24, 32, 128)  512
['conv2d_16[0][0]']
 ormalization)

 conv2d_17 (Conv2D)             (None, 24, 32, 128)  147584
['batch_normalization_14[0][0]']
```

```
 conv2d_transpose_3 (Conv2DTran  (None, 48, 64, 64)  73792
['conv2d_17[0][0]']
 spose)

 concatenate_3 (Concatenate)    (None, 48, 64, 128)  0
['conv2d_transpose_3[0][0]',

'batch_normalization_3[0][0]']

 conv2d_18 (Conv2D)             (None, 48, 64, 64)   73792
['concatenate_3[0][0]']

 batch_normalization_15 (BatchN  (None, 48, 64, 64)  256
['conv2d_18[0][0]']
 ormalization)

 conv2d_19 (Conv2D)             (None, 48, 64, 64)   36928
['batch_normalization_15[0][0]']

 conv2d_transpose_4 (Conv2DTran  (None, 96, 128, 32)  165920
['conv2d_19[0][0]']
 spose)

 concatenate_4 (Concatenate)    (None, 96, 128, 64)  0
['conv2d_transpose_4[0][0]',

'batch_normalization_1[0][0]']

 conv2d_20 (Conv2D)             (None, 96, 128, 32)  165920
['concatenate_4[0][0]']

 batch_normalization_16 (BatchN  (None, 96, 128, 32)  128
['conv2d_20[0][0]']
 ormalization)

 conv2d_21 (Conv2D)             (None, 96, 128, 32)  82976
['batch_normalization_16[0][0]']

 conv2d_22 (Conv2D)             (None, 96, 128, 32)  50208
['conv2d_21[0][0]']

 conv2d_23 (Conv2D)             (None, 96, 128, 12)  396
['conv2d_22[0][0]']
```

Fig. 8a : Model Calculations

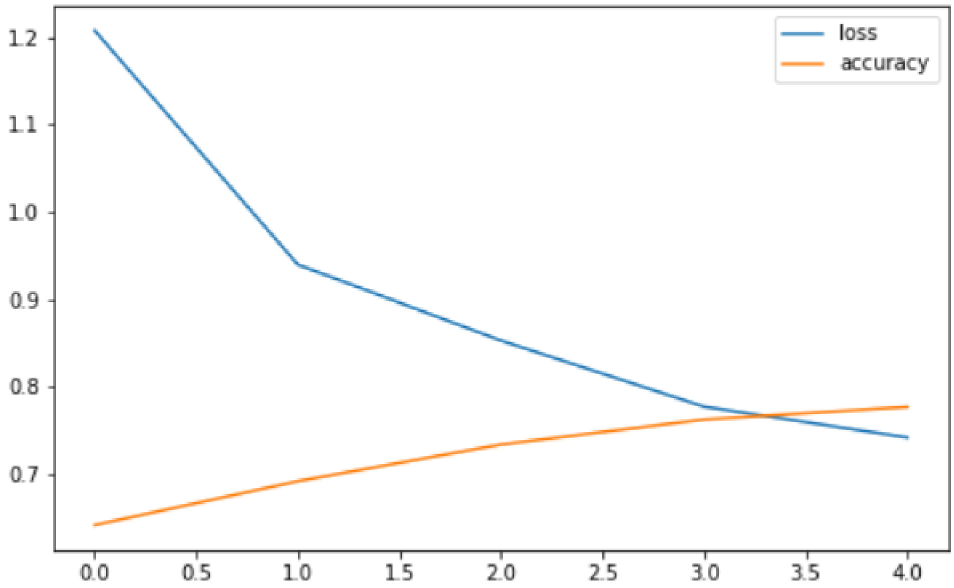


Fig. 8b : Training Loss and Accuracy Graph with Epochs

Metrics and Results:

The following code computes the class wise metrics as shown in the Fig. 9.

```
true_masks,predicted_masks = [], []
pred_mask = unet.predict(Train_Images)
pred_mask = tf.expand_dims(tf.argmax(pred_mask,axis=-1), axis = -1)
predicted_masks.extend(pred_mask)
predicted_masks = np.array(predicted_masks)

def evaluate(true_masks, predicted_masks, n_classes, smooth = 1e-6):

  class_wise_true_positives, class_wise_true_negatives = [],[]
  class_wise_false_positives, class_wise_false_negatives = [],[]
  class_wise_precisions, class_wise_recalls = [],[]
  class_wise_specificities, class_wise_ious = [],[]
  class_wise_tdrs, class_wise_f1_scores = [],[]
  classes = []
  for clas in range(n_classes):
    true_positives, true_negatives, false_positives, false_negatives =
0,0,0,0
    precisions, recalls, specificities, ious, f1_scores, tdrs = 0,0,0,0
,0,0
    number_of_masks = true_masks.shape[0]

    for mask_id in range(number_of_masks):
```

```
      true_positive = np.sum(np.logical_and(true_masks[mask_id]==clas,
predicted_masks[mask_id]==clas))
      true_negative = np.sum(np.logical_and(true_masks[mask_id]!=clas,
predicted_masks[mask_id]!=clas))
      false_positive = np.sum(np.logical_and(true_masks[mask_id]!=clas,
 predicted_masks[mask_id]==clas))
      false_negative = np.sum(np.logical_and(true_masks[mask_id]==clas,
 predicted_masks[mask_id]!=clas))

      true_positives += true_positive
      true_negatives += true_negative
      false_positives += false_positive
      false_negatives += false_negative
    recall = round(true_positives/(true_positives + false_negatives + s
mooth), 2)
    precision = round(true_positives/(true_positives + false_positives
+ smooth), 2)
    specificity = round(true_negatives/(true_negatives + false_positive
s + smooth), 2)
    tdr = round((1 - (false_negatives/(true_positives + false_negatives
 + smooth))), 2)
    iou = round(true_positives/(true_positives + false_negatives + fals
e_positives + smooth), 2)
    f1_score = round((2 * precision * recall)/(precision + recall + smo
oth), 2)

    class_wise_true_positives.append(true_positives)
    class_wise_true_negatives.append(true_negatives)
    class_wise_false_positives.append(false_positives)
    class_wise_false_negatives.append(false_negatives)
    class_wise_recalls.append(recall)
    class_wise_precisions.append(precision)
    class_wise_specificities.append(specificity)
    class_wise_ious.append(iou)
    class_wise_tdrs.append(tdr)
    class_wise_f1_scores.append(f1_score)
    classes.append("Class " + str(clas+1))

  total_true_positives = np.sum(class_wise_true_positives)
  total_true_negatives = np.sum(class_wise_true_negatives)
  total_false_positives = np.sum(class_wise_false_positives)
  total_false_negatives = np.sum(class_wise_false_negatives)
  mean_recall = round(np.average(np.array(class_wise_recalls)), 2)
  mean_precision = round(np.average(np.array(class_wise_precisions)), 2
)
  mean_specificity = round(np.average(np.array(class_wise_specificities
)), 2)
  mean_iou = round(np.average(np.array(class_wise_ious)), 2)
```

```
  mean_tdr = round(np.average(np.array(class_wise_tdrs)), 2)
  mean_f1_score = round(np.average(np.array(class_wise_f1_scores)), 2)


  class_wise_evaluations = {"Class": classes,
                            "True Positive Pixels": class_wise_true_p
ositives,
                            "True Negative Pixels": class_wise_true_n
egatives,
                            "False Positive Pixels": class_wise_false
_positives,
                            "False Negative Pixels": class_wise_false
_negatives,
                            "Recall": class_wise_recalls,
                            "Precision": class_wise_precisions,
                            "Specificity": class_wise_specificities,
                            "IoU": class_wise_ious,
                            "TDR": class_wise_tdrs,
                            "F1-Score": class_wise_f1_scores}

  overall_evaluations = {"Class": "All Classes",
                         "True Positive Pixels": total_true_positives,
                         "True Negative Pixels": total_true_negatives,
                         "False Positive Pixels": total_false_positives,
                         "False Negative Pixels": total_false_negatives,
                         "Recall": mean_recall,
                         "Precision": mean_precision,
                         "Specificity": mean_specificity,
                         "IoU": mean_iou,
                         "TDR": mean_tdr,
                         "F1-Score": mean_f1_score}

  evaluations = {"Overall Evaluations": overall_evaluations,
                 "Class-wise Evaluations": class_wise_evaluations}

  metrics=["Recall", "Precision", "Specificity", "IoU", "TDR", "F1 Scor
e"]

  return evaluations, metrics

evaulations, metrics = evaluate(Train_Masks, predicted_masks, 12)

Class_Wise = evaulations['Class-wise Evaluations']
print('Classes:',Class_Wise['Class'])
print('TP:',Class_Wise['True Positive Pixels'])
print('TN:',Class_Wise['True Negative Pixels'])
print('FP:',Class_Wise['False Positive Pixels'])
print('FN:',Class_Wise['False Negative Pixels'])
```

```
print('Precision:',Class_Wise['Precision'])
print('Recall:',Class_Wise['Recall'])
print('IOU:',Class_Wise['IoU'])
print('F_score:',Class_Wise['F1-Score'])
```

```
Classes: ['Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class
6', 'Class 7', 'Class 8', 'Class 9', 'Class 10', 'Class 11', 'Class
12']
TP: [627730, 901301, 0, 1425425, 1666, 4843, 0, 0, 7439, 0, 0, 411]
TN: [3692141, 2742005, 4465239, 2427326, 4213151, 4070181, 4457031,
4458894, 4234640, 4480834, 4496576, 4327757]
FP: [57537, 719091, 0, 655142, 94308, 988, 0, 0, 10477, 0, 0, 3338]
FN: [132288, 147299, 44457, 1803, 200571, 433684, 52665, 50802, 257140,
28862, 13120, 178190]
Precision: [0.92, 0.56, 0.0, 0.69, 0.02, 0.83, 0.0, 0.0, 0.42, 0.0,
0.0, 0.11]
Recall: [0.83, 0.86, 0.0, 1.0, 0.01, 0.01, 0.0, 0.0, 0.03, 0.0, 0.0,
0.0]
IOU: [0.77, 0.51, 0.0, 0.68, 0.01, 0.01, 0.0, 0.0, 0.03, 0.0, 0.0, 0.0]
 F_score: [0.87, 0.68, 0.0, 0.82, 0.01, 0.02, 0.0, 0.0, 0.06, 0.0, 0.0,
0.0]
```

Fig. 9 : Unet Metrics

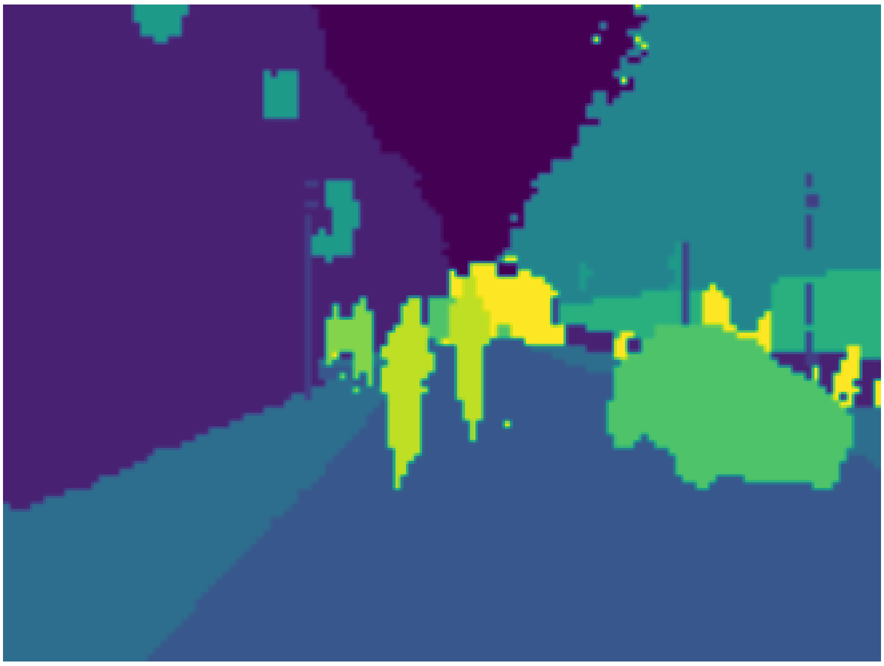
The true masks, predicted masks and images are shown as results in Fig 9A for which the code is as following:

```
def display(l_list):

  title = ["Input Image", "True Mask", "Predicted Mask"]

  for i in range(len(l_list)):
    plt.figure(figsize=(20, 20))

    plt.subplot(1, len(l_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(l_list[i]))
    plt.axis('off')

    plt.show()
def create_mask(pred_mask):
    pred_mask = tf.argmax(pred_mask, axis=-1)
```

```
    pred_mask = pred_mask[..., tf.newaxis]
    return pred_mask[0]
for i in range(10):
    index = np.random.randint(Test_Images.shape[0])

    input_image = Test_Images[index]
    true_mask = Test_Masks[index]

    pred_mask = unet(tf.expand_dims(input_image, axis=0))
    pred_mask = create_mask(pred_mask)

    display([input_image, true_mask, pred_mask])
```

Input Image

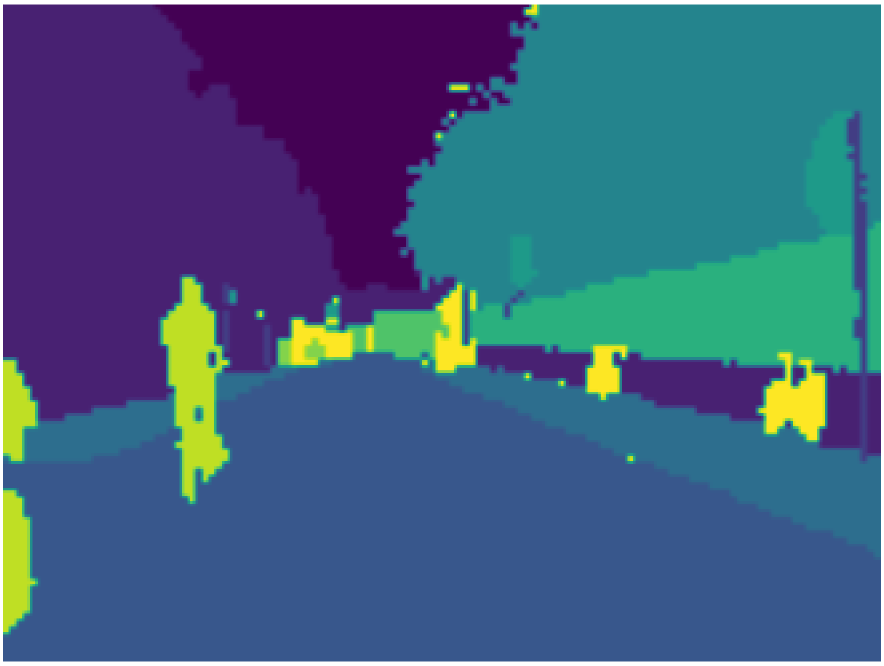
True Mask



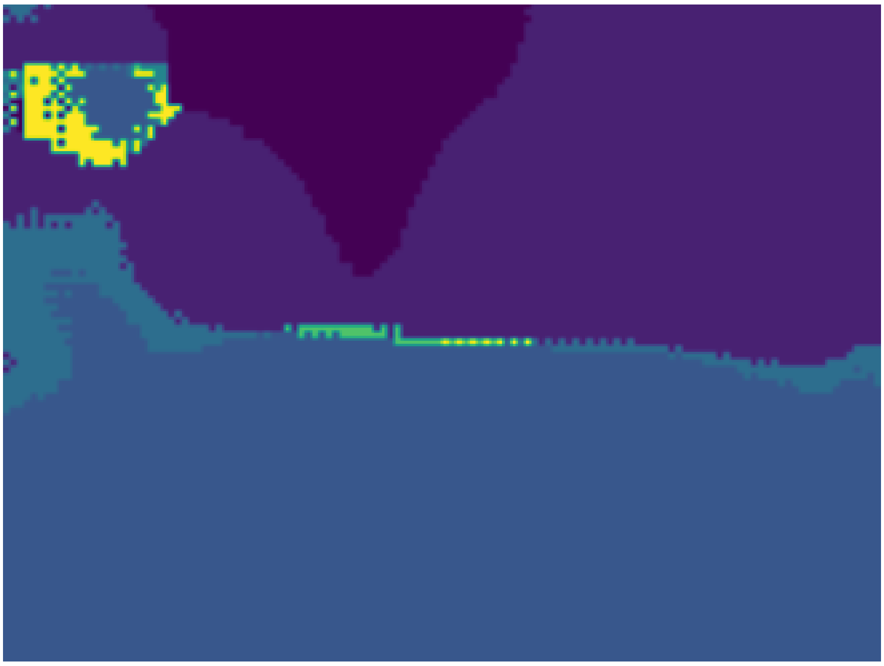
Predicted Mask

Input Image

True Mask

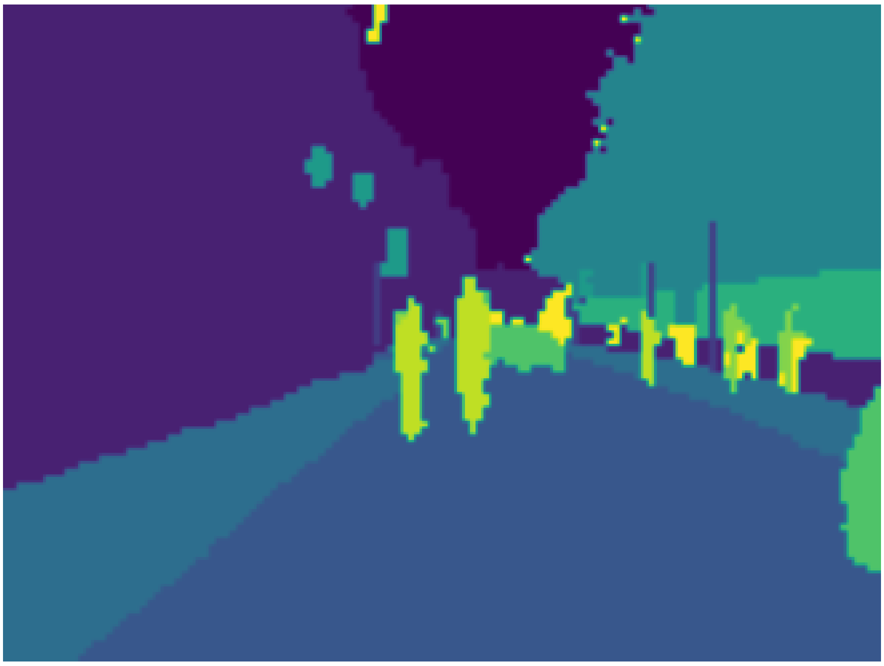


Predicted Mask



Input Image

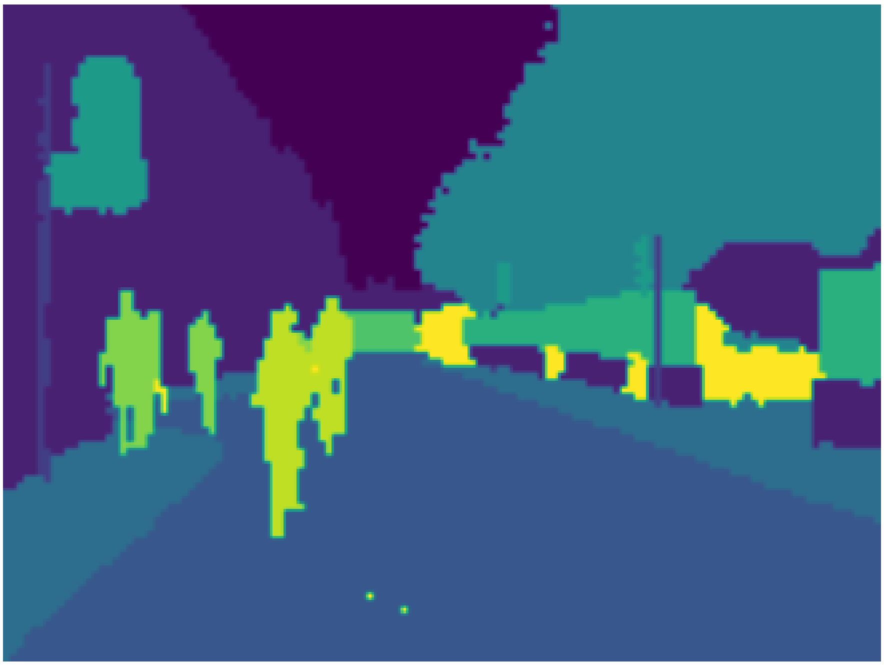
True Mask



Predicted Mask

Input Image

True Mask



Predicted Mask

Input Image

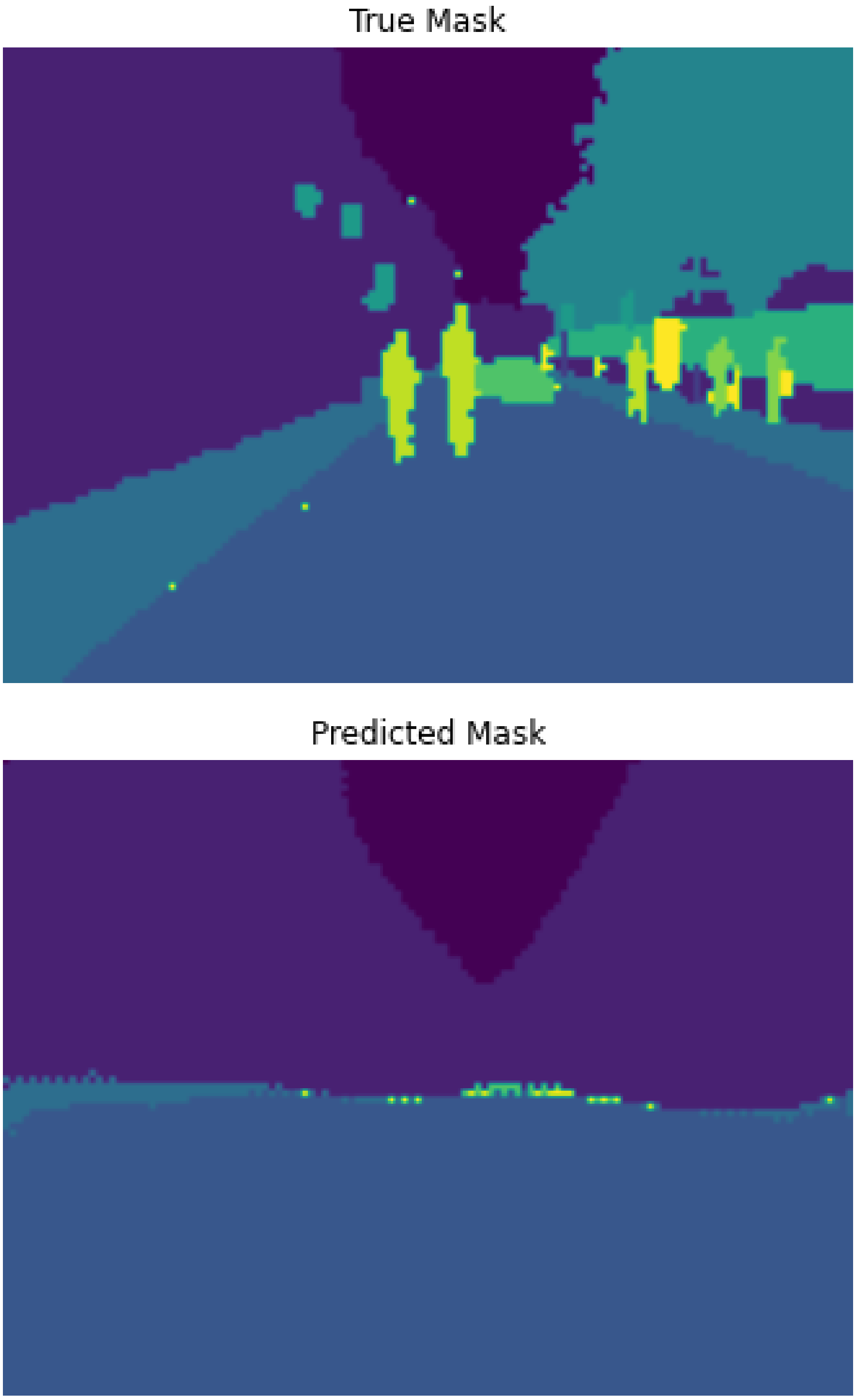


Fig. 9A : Images, True Masks and Predicted Masks

Second Model of Unet:

A second model was created using same conv blocks and deconv blocks. The filter size in this model is fixed i.e., 3 until last layer where a 1x1 filter is used for output Here there are 5 conv layers and 5 deconv layers so the size has been reduced. I have also added regularizers in final conv layers and weights are initialized in them using Random Normal. Fig. 10 shows the architecture diagram and Fig. 11 shows all step wise calculations including step wise parameters and output shapes. Fig. 12a shows loss and accuracy curves and the class wise results are shown in Fig. 12b. All codes until metrics evaluation are given below:

```
def unet2(input_size, num_filters, num_classes):
  inputs = Input(input_size)
  conv_block_1 = conv_block(inputs =inputs, num_filters = num_filters*1, filter_size = 3, dropout_p=0)
  conv_block_2 = conv_block(inputs = conv_block_1[0],num_filters = num_filters*2, filter_size = 3, dropout_p=0)
  conv_block_3 = conv_block(inputs = conv_block_2[0],num_filters = num_filters*4, filter_size = 3, dropout_p=0)
  conv_block_4 = conv_block(inputs = conv_block_3[0],num_filters = num_filters*8, filter_size = 3, dropout_p=0.3)
  conv_block_5 = conv_block(inputs = conv_block_4[0],num_filters = num_filters*32, filter_size = 3, dropout_p=0.3, max_pool = False)
  deconv_block_6 = upsampling_block(conv_block_5[0], conv_block_4[1], num_filters * 16, filter_size = 3)
  deconv_block_7 = upsampling_block(deconv_block_6, conv_block_3[1], num_filters * 8, filter_size = 3)
  deconv_block_8 = upsampling_block(deconv_block_7, conv_block_2[1], num_filters * 4, filter_size = 3)
  deconv_block_9 = upsampling_block(deconv_block_8, conv_block_1[1], num_filters * 2, filter_size = 3)
  conv2d = Conv2D(num_filters,(3, 3),activation="relu", padding="same", kernel_initializer="RandomNormal", kernel_regularizer = 'l2')(deconv_block_9)
  # Add a Conv2D layer with num_classes filter, kernel size of 1 and a 'same' padding
  outputs = Conv2D(num_classes, 1, padding="same")(conv2d)

  model = tf.keras.Model(inputs=inputs, outputs=outputs)

  return model
```

```
unet2 = unet2(input_size=Train_Images[0].shape, num_filters= 32, num_cl
asses=12)
 true_masks,predicted_masks2 = [], []

pred_mask = unet2.predict(Train_Images)
pred_mask2 = tf.expand_dims(tf.argmax(pred_mask,axis=-1), axis = -1)
predicted_masks2.extend(pred_mask2)
predicted_masks2 = np.array(predicted_masks2)

evaulations2, metrics = evaluate(Train_Masks, predicted_masks2, 12)
Class_Wise2 = evaulations2['Class-wise Evaluations']
print('Classes:',Class_Wise2['Class'])
print('TP:',Class_Wise2['True Positive Pixels'])
print('TN:',Class_Wise2['True Negative Pixels'])
print('FP:',Class_Wise2['False Positive Pixels'])
print('FN:',Class_Wise2['False Negative Pixels'])
print('Precision:',Class_Wise2['Precision'])
print('Recall:',Class_Wise2['Recall'])
print('IOU:',Class_Wise2['IoU'])
print('F_score:',Class_Wise2['F1-Score'])
```

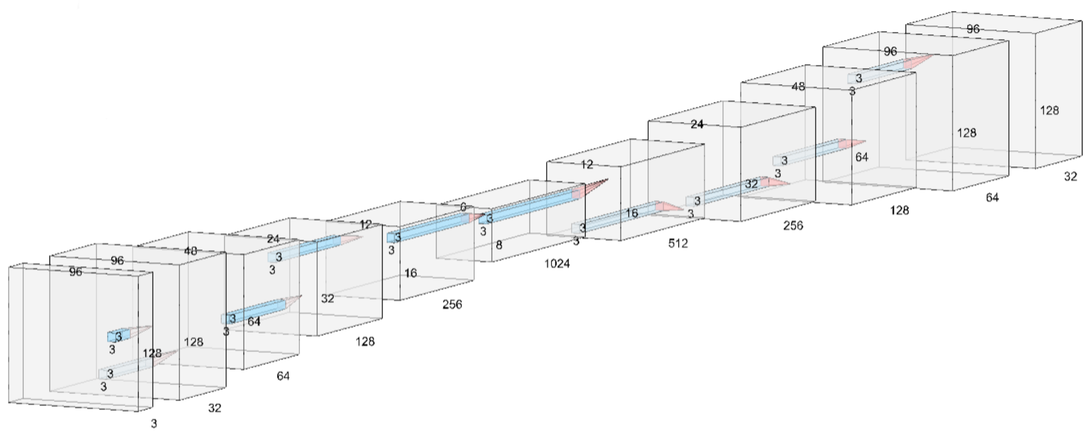


Fig. 10 : Unet 2nd Model

```
Model: "model_1"
```

```
________________________________________________________________________________
______________________________
 Layer (type)                Output Shape         Param #
Connected to
================================================================================
==============================
 input_2 (InputLayer)         [(None, 96, 128, 3)  0            []
                              ]

 conv2d_24 (Conv2D)           (None, 96, 128, 32)  896
['input_2[0][0]']

 batch_normalization_17 (BatchN  (None, 96, 128, 32)  128
['conv2d_24[0][0]']
 ormalization)

 conv2d_25 (Conv2D)           (None, 96, 128, 32)  9248
['batch_normalization_17[0][0]']

 batch_normalization_18 (BatchN  (None, 96, 128, 32)  128
['conv2d_25[0][0]']
 ormalization)

 max_pooling2d_5 (MaxPooling2D)  (None, 48, 64, 32)  0
['batch_normalization_18[0][0]']

 conv2d_26 (Conv2D)           (None, 48, 64, 64)   18496
['max_pooling2d_5[0][0]']

 batch_normalization_19 (BatchN  (None, 48, 64, 64)  256
['conv2d_26[0][0]']
 ormalization)

 conv2d_27 (Conv2D)           (None, 48, 64, 64)   36928
['batch_normalization_19[0][0]']

 batch_normalization_20 (BatchN  (None, 48, 64, 64)  256
['conv2d_27[0][0]']
 ormalization)

 max_pooling2d_6 (MaxPooling2D)  (None, 24, 32, 64)  0
['batch_normalization_20[0][0]']

 conv2d_28 (Conv2D)           (None, 24, 32, 128)  73856
['max_pooling2d_6[0][0]']

 batch_normalization_21 (BatchN  (None, 24, 32, 128)  512
['conv2d_28[0][0]']
 ormalization)

 conv2d_29 (Conv2D)           (None, 24, 32, 128)  147584
['batch_normalization_21[0][0]']

 batch_normalization_22 (BatchN  (None, 24, 32, 128)  512
['conv2d_29[0][0]']
 ormalization)
```

```
 max_pooling2d_7 (MaxPooling2D)  (None, 12, 16, 128)  0
['batch_normalization_22[0][0]']

 conv2d_30 (Conv2D)             (None, 12, 16, 256)  295168
['max_pooling2d_7[0][0]']

 batch_normalization_23 (BatchN  (None, 12, 16, 256)  1024
['conv2d_30[0][0]']
 ormalization)

 conv2d_31 (Conv2D)             (None, 12, 16, 256)  590080
['batch_normalization_23[0][0]']

 batch_normalization_24 (BatchN  (None, 12, 16, 256)  1024
['conv2d_31[0][0]']
 ormalization)

 dropout_2 (Dropout)            (None, 12, 16, 256)  0
['batch_normalization_24[0][0]']

 max_pooling2d_8 (MaxPooling2D)  (None, 6, 8, 256)   0
['dropout_2[0][0]']

 conv2d_32 (Conv2D)             (None, 6, 8, 1024)   2360320
['max_pooling2d_8[0][0]']

 batch_normalization_25 (BatchN  (None, 6, 8, 1024)  4096
['conv2d_32[0][0]']
 ormalization)

 conv2d_33 (Conv2D)             (None, 6, 8, 1024)   9438208
['batch_normalization_25[0][0]']

 batch_normalization_26 (BatchN  (None, 6, 8, 1024)  4096
['conv2d_33[0][0]']
 ormalization)

 dropout_3 (Dropout)            (None, 6, 8, 1024)   0
['batch_normalization_26[0][0]']

 conv2d_transpose_5 (Conv2DTran  (None, 12, 16, 512)  4719104
['dropout_3[0][0]']
 spose)

 concatenate_5 (Concatenate)    (None, 12, 16, 768)  0
['conv2d_transpose_5[0][0]',

'dropout_2[0][0]']

 conv2d_34 (Conv2D)             (None, 12, 16, 512)  3539456
['concatenate_5[0][0]']

 batch_normalization_27 (BatchN  (None, 12, 16, 512)  2048
['conv2d_34[0][0]']
 ormalization)

 conv2d_35 (Conv2D)             (None, 12, 16, 512)  2359808
['batch_normalization_27[0][0]']
```

```
 conv2d_transpose_6 (Conv2DTran  (None, 24, 32, 256)  1179904
['conv2d_35[0][0]']
 spose)

 concatenate_6 (Concatenate)    (None, 24, 32, 384)  0
['conv2d_transpose_6[0][0]',

'batch_normalization_22[0][0]']

 conv2d_36 (Conv2D)             (None, 24, 32, 256)  884992
['concatenate_6[0][0]']

 batch_normalization_28 (BatchN  (None, 24, 32, 256)  1024
['conv2d_36[0][0]']
 ormalization)

 conv2d_37 (Conv2D)             (None, 24, 32, 256)  590080
['batch_normalization_28[0][0]']

 conv2d_transpose_7 (Conv2DTran  (None, 48, 64, 128)  295040
['conv2d_37[0][0]']
 spose)

 concatenate_7 (Concatenate)    (None, 48, 64, 192)  0
['conv2d_transpose_7[0][0]',

'batch_normalization_20[0][0]']

 conv2d_38 (Conv2D)             (None, 48, 64, 128)  221312
['concatenate_7[0][0]']

 batch_normalization_29 (BatchN  (None, 48, 64, 128)  512
['conv2d_38[0][0]']
 ormalization)

 conv2d_39 (Conv2D)             (None, 48, 64, 128)  147584
['batch_normalization_29[0][0]']

 conv2d_transpose_8 (Conv2DTran  (None, 96, 128, 64)  73792
['conv2d_39[0][0]']
 spose)

 concatenate_8 (Concatenate)    (None, 96, 128, 96)  0
['conv2d_transpose_8[0][0]',

'batch_normalization_18[0][0]']

 conv2d_40 (Conv2D)             (None, 96, 128, 64)  55360
['concatenate_8[0][0]']

 batch_normalization_30 (BatchN  (None, 96, 128, 64)  256
['conv2d_40[0][0]']
 ormalization)

 conv2d_41 (Conv2D)             (None, 96, 128, 64)  36928
['batch_normalization_30[0][0]']
```

```
 conv2d_42 (Conv2D)              (None, 96, 128, 32)  18464
['conv2d_41[0][0]']

 conv2d_43 (Conv2D)              (None, 96, 128, 12)  396
['conv2d_42[0][0]']
```

Fig. 11 : Unet 2nd Model Calculations

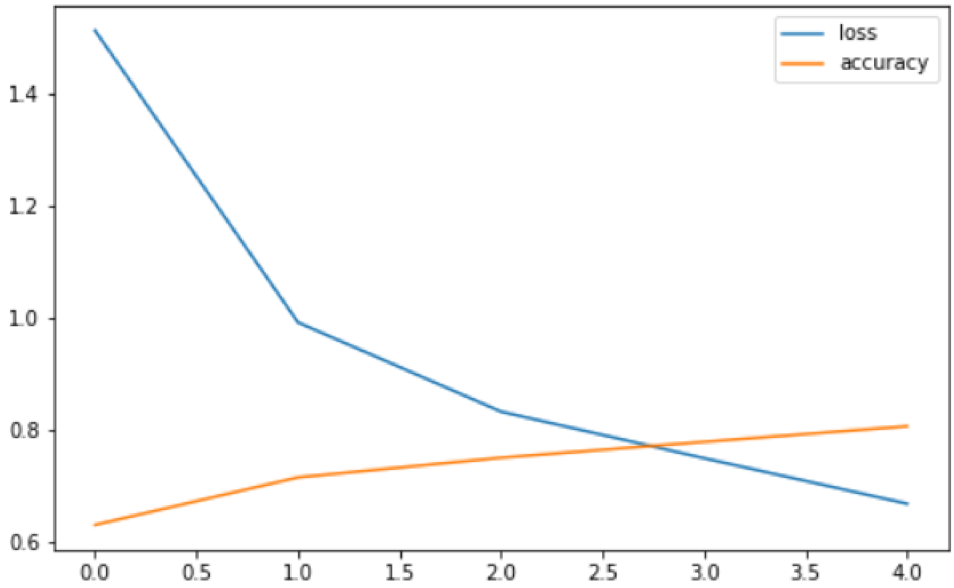


Fig. 12a : Unet 2nd Model Training Loss and Accuracy

```
Classes: ['Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class
6', 'Class 7', 'Class 8', 'Class 9', 'Class 10', 'Class 11', 'Class
12']
TP: [714329, 979187, 0, 738783, 18558, 14981, 0, 0, 3785, 0, 0, 3]
TN: [3544489, 1893984, 4465239, 2957503, 4165340, 4070864, 4457031,
4458894, 4244737, 4480834, 4496576, 4331095]
FP: [205189, 1567112, 0, 124965, 142119, 305, 0, 0, 380, 0, 0, 0]
FN: [45689, 69413, 44457, 688445, 183679, 423546, 52665, 50802, 260794,
28862, 13120, 178598]
Precision: [0.78, 0.38, 0.0, 0.86, 0.12, 0.98, 0.0, 0.0, 0.91, 0.0,
0.0, 1.0]
Recall: [0.94, 0.93, 0.0, 0.52, 0.09, 0.03, 0.0, 0.0, 0.01, 0.0, 0.0,
0.0]
IOU: [0.74, 0.37, 0.0, 0.48, 0.05, 0.03, 0.0, 0.0, 0.01, 0.0, 0.0, 0.0]
 F_score: [0.85, 0.54, 0.0, 0.65, 0.1, 0.06, 0.0, 0.0, 0.02, 0.0, 0.0,
0.0]
```

Fig. 12b : Unet 2nd Model Metrics

Ensemble:

I have used averaging method for ensemble here where I have taken the class wise IoU values from both models and taken their averages to compute IoU values for ensemble. It is because both models have performed good on this dataset so I have given them equal weightages by which their averages will be calculated.

The code is below and the results are shown in Fig. 13.

```
import math
def ensemble(Model_1_IoU_results, Model_2_IoU_results,num_classes):
  ensemble_IoU = []
  for i in range(num_classes):
    sum = Model_1_IoU_results[i] + Model_2_IoU_results[i]
    average = sum/2
    ensemble_IoU.append(average)
  return ensemble_IoU
ensemble_IoU = ensemble(Class_Wise['IoU'],Class_Wise2['IoU'],num_classe
s = 12)

final = pd.DataFrame([ensemble_IoU],columns = Class_Wise['Class'], inde
x = ['ensemble'] )
final
```

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11	Class 12
ensemble	0.74	0.37	0.0	0.48	0.05	0.03	0.0	0.0	0.01	0.0	0.0	0.0

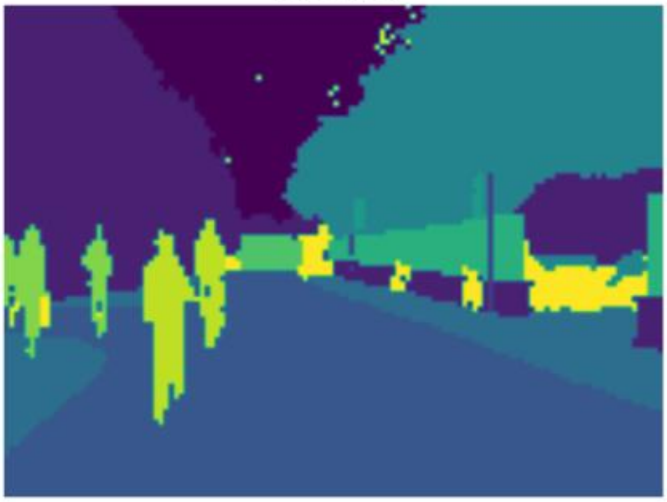
Fig. 13 : Ensemble IoU values

Annexure:

Results of 2nd Unet Model:

Input Image

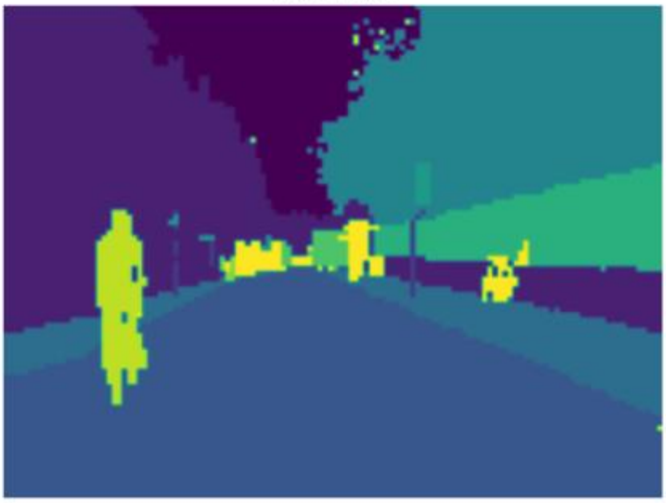
True Mask



Predicted Mask

Input Image

True Mask



Predicted Mask