Problem Statement:

Given Satellite imagery, extract road features from it

Dataset Location:

Navigate to https://www.kaggle.com/insaff/massachusetts-roads-dataset

About Dataset:

The Data is split into two categories Training and Testing. Each input image is in RGB format, and its associated label is in binary format. Label with pixel value 1 represents road element and with pixel value 0 represent non-road elements.

Approach:

Tried with machine learning and CNN

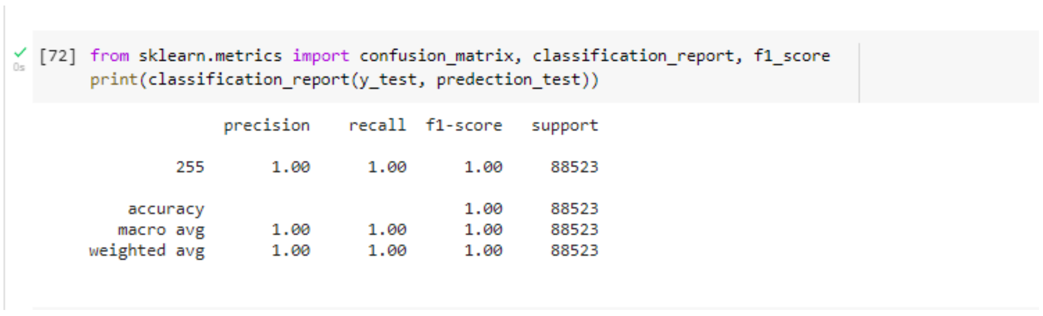
1. Machine Learning-Random Forest

- Implemented using random forest
- Used Gabor filter to extract the features
- Generated other features like- Robert, Sobel, Scharr, prewitt, guassian,variance

Steps:

1. Read training images and extract features
2. Read labeled images (MASKS) and create new Dataframe
3. Create gabor filter and other features
4. Define the random forest model and fit the model using training data
5. Check the accuracy of the model
6. Save the model for future use
7. Make prediction of test images

Using this approach, the accuracy we are getting is 100%, which is wired.
And f1 score and precession are also 100%. Need to trouble shoot.
File- TOM_TOM_ASSIGNMENT_JALPESH_RF.ipynb



```
[72] from sklearn.metrics import confusion_matrix, classification_report, f1_score
     print(classification_report(y_test, predection_test))

              precision    recall  f1-score   support

         255       1.00      1.00      1.00     88523

    accuracy                           1.00     88523
   macro avg       1.00      1.00      1.00     88523
weighted avg       1.00      1.00      1.00     88523
```

2. Deep Learning- UNet

- Implemented Unet with accuracy and Jaccard coefficient metrics
- File
    - TOM_TOM__ASSIGMENT_JALPESH_Standard_Model.ipynb
    - train_model.py
    - unet_model.py
    - TOM_TOM__ASSIGMENT_JALPESH_jacard_model.ipynb

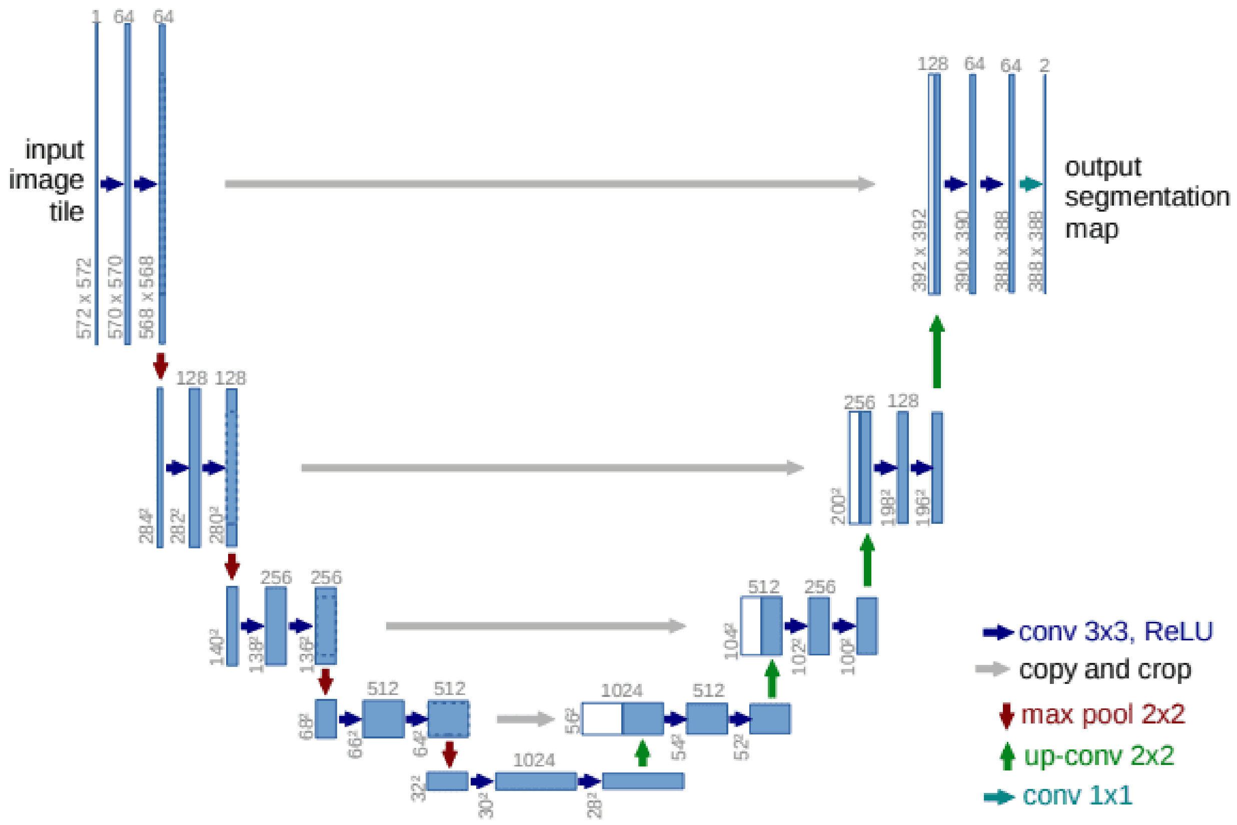
Steps:

1. Data Preprocessing

- Processing Input images- Satellite images
- Processing output Images- Masks Images- considering only one channel
- Converting into array
- Normalizing the data

2. Create UNet Model

- UNet Model Implemented for segmentation
- Input shape is 236,236,3
- Output shape is 236,236,1



The UNet architecture was introduced for Biomedical Image segmentation by Olag Ronneberger et al. The introduced architecture had two main parts that were encoder and decoder.

Encoder (left side): It consists of the repeated application of two 3x3 convolutions. Each conv is followed by a ReLU and batch normalization. Then a 2x2 max pooling operation is applied to reduce the spatial dimensions. Again, at each down sampling step, we double the number of feature channels, while we cut in half the spatial dimensions.

Decoder path (right side): Every step in the expansive path consists of an up sampling of the feature map followed by a 2x2 transpose convolution, which halves the number of feature channels. We also have a concatenation with the corresponding feature map from the contracting path, and usually a 3x3 convolutional (each followed by a ReLU). At the final layer, a 1x1 convolution is used to map the channels to the desired number of classes.

3. Model Check point

- Early stopping – patience =3 on val_loss

4. Instantiating the model

- Optimizer: Adam
- Loss: binary_crossentropy
- Metrics= accuracy

5. Run the model on training and validation set

- Epochs:10
- Batch size:16
- callbacks= early stopping

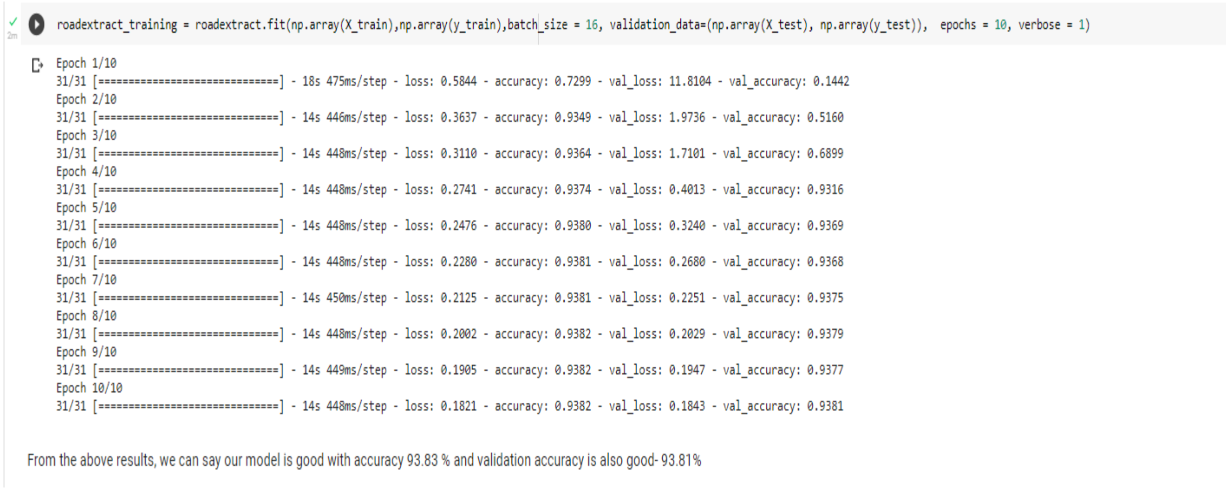
6. Evaluate Model

Initially, I ran the model on normalized data. It was giving good accuracy on training and validation set. However, In the predicated image included lot of noise.

Accuracy – 93.81%

IOU -13%

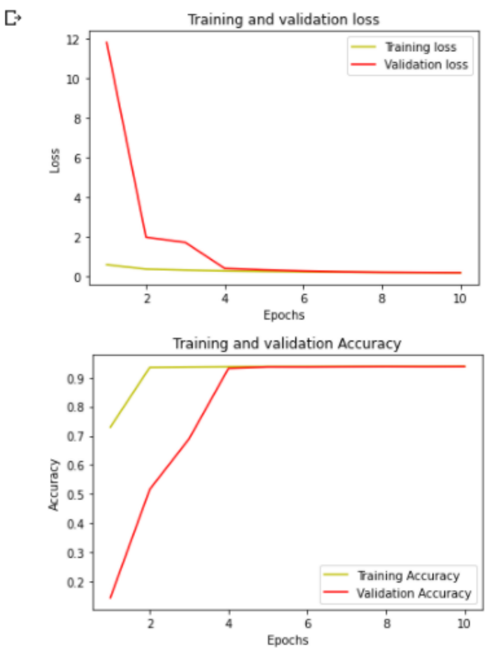
Normalized data-



```
roadextract_training = roadextract.fit(np.array(X_train),np.array(y_train),batch_size = 16, validation_data=(np.array(X_test), np.array(y_test)),  epochs = 10, verbose = 1)
```

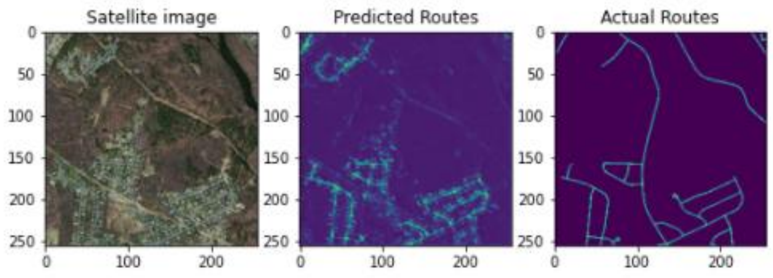
```
Epoch 1/10
31/31 [==============================] - 18s 475ms/step - loss: 0.5844 - accuracy: 0.7299 - val_loss: 11.8104 - val_accuracy: 0.1442
Epoch 2/10
31/31 [==============================] - 14s 446ms/step - loss: 0.3637 - accuracy: 0.9349 - val_loss: 1.9736 - val_accuracy: 0.5160
Epoch 3/10
31/31 [==============================] - 14s 448ms/step - loss: 0.3110 - accuracy: 0.9364 - val_loss: 1.7101 - val_accuracy: 0.6899
Epoch 4/10
31/31 [==============================] - 14s 448ms/step - loss: 0.2741 - accuracy: 0.9374 - val_loss: 0.4013 - val_accuracy: 0.9316
Epoch 5/10
31/31 [==============================] - 14s 448ms/step - loss: 0.2476 - accuracy: 0.9380 - val_loss: 0.3240 - val_accuracy: 0.9369
Epoch 6/10
31/31 [==============================] - 14s 448ms/step - loss: 0.2280 - accuracy: 0.9381 - val_loss: 0.2680 - val_accuracy: 0.9368
Epoch 7/10
31/31 [==============================] - 14s 450ms/step - loss: 0.2125 - accuracy: 0.9381 - val_loss: 0.2251 - val_accuracy: 0.9375
Epoch 8/10
31/31 [==============================] - 14s 448ms/step - loss: 0.2002 - accuracy: 0.9382 - val_loss: 0.2029 - val_accuracy: 0.9379
Epoch 9/10
31/31 [==============================] - 14s 449ms/step - loss: 0.1905 - accuracy: 0.9382 - val_loss: 0.1947 - val_accuracy: 0.9377
Epoch 10/10
31/31 [==============================] - 14s 448ms/step - loss: 0.1821 - accuracy: 0.9382 - val_loss: 0.1843 - val_accuracy: 0.9381
```

From the above results, we can say our model is good with accuracy 93.83 % and validation accuracy is also good- 93.81%

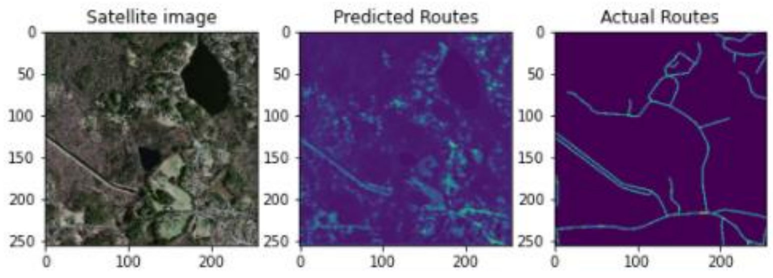


Training data set

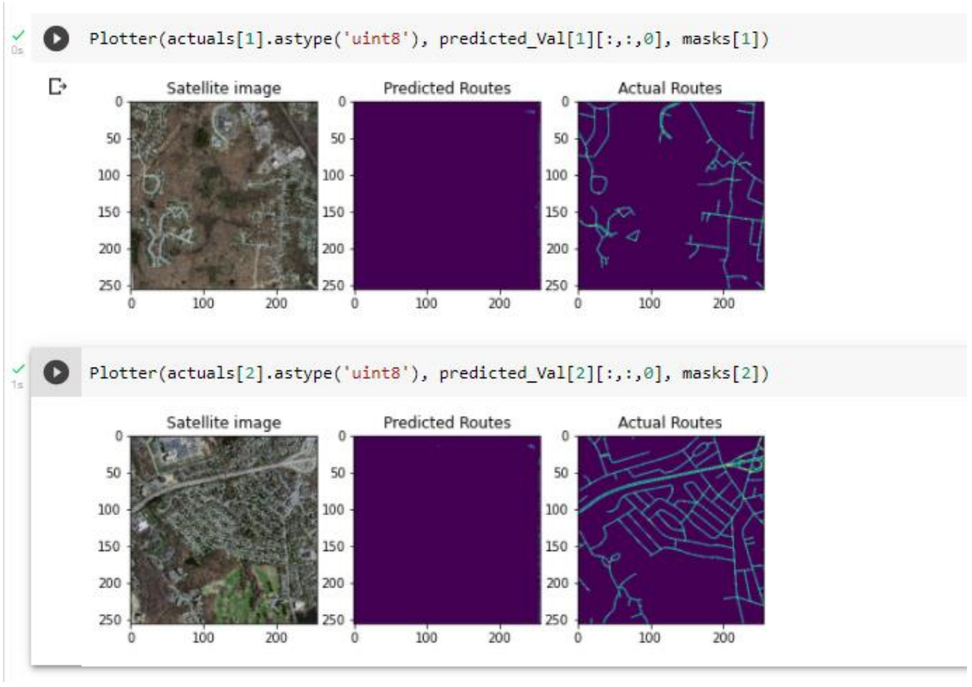
```
[64] Plotter(actuals[1], predicted[1][:,:,0], masks[1])
```



```
Plotter(actuals[2], predicted[2][:,:,0], masks[2])
```



Test dataset:

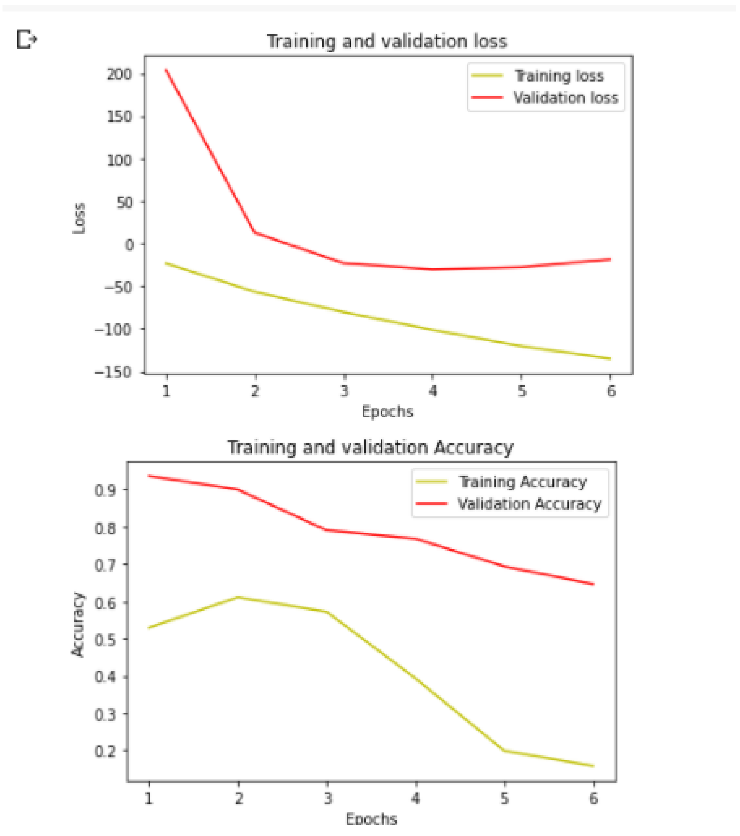


```
Plotter(actuals[1].astype('uint8'), predicted_Val[1][:,:,0], masks[1])
```

```
Plotter(actuals[2].astype('uint8'), predicted_Val[2][:,:,0], masks[2])
```

I tried without normalization and predicted images were looking good as compared to normalized data.

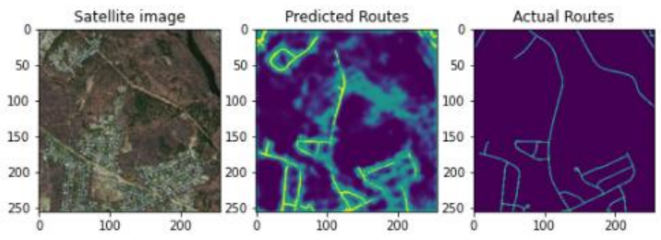
- Accuracy we got is 64.64% on test data



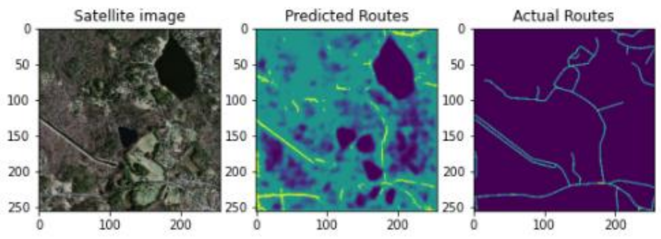
Training dataset results:

```
[111] predicted, actuals, masks = predict(TrainingObj, roadextract)
```

```
[113] Plotter(actuals[1].astype('uint8'), predicted[1][:,:,0], masks[1])
```



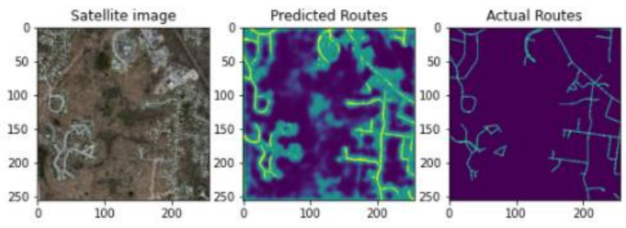
```
Plotter(actuals[2].astype('uint8'), predicted[2][:,:,0], masks[2])
```



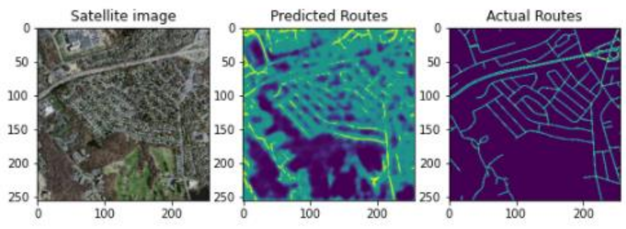
Validation dataset results:

```
[120] predicted_Val, actuals, masks = predict(ValidationObj, roadextract)
```

```
[121] Plotter(actuals[1].astype('uint8'), predicted_Val[1][:,:,0], masks[1])
```



```
Plotter(actuals[2].astype('uint8'), predicted_Val[2][:,:,0], masks[2])
```



7. Alternate Approach

- Tried with Jacard Metrics
- Accuracy we got with this metrics is only 5.55%
- IOU score is 1.1%

-

8. Conclusion

- From the results, we can say model created is underfit model. Training accuracy is less as compared to test accuracy.
- We can enhance the model using different activation function.
- Also, we can enhance the model by implementing GAN