SATELLITE IMAGERY USING DEEP LEARNING

TEAM MEMBER NAMES

VENKATA NIKITH PULGAM

Student Id:U00885957

NAGA SRI VANDANA PARIMALA SURE

Student Id:U00885835

Introduction

Satellite imagery has become an increasingly important source of data for a wide range of applications, including agriculture, environmental monitoring, urban planning, and disaster management. Deep learning techniques can be applied to satellite imagery to extract meaningful information and patterns from the data.

Deep learning is a subfield of machine learning that involves training artificial neural networks to learn from large amounts of data. In the context of satellite imagery, deep learning can be used to classify land cover, detect changes in land use over time, identify objects such as buildings and roads, and predict environmental variables such as soil moisture and vegetation density.

The first step in using deep learning for satellite imagery is to acquire and pre-process the data. This may involve downloading the imagery from satellite data archives, applying radiometric and atmospheric corrections to the data, and cropping the imagery to the area of interest.

Once the data has been preprocessed, the next step is to train a deep learning model. This involves selecting an appropriate neural network architecture and training it on a labeled dataset. The labeled dataset may include examples of different land cover types, object classes, or

environmental variables, along with corresponding labels indicating the correct classification or prediction for each example.

There are a variety of neural network architectures that can be used for satellite imagery analysis, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention-based models. CNNs are commonly used for tasks such as land cover classification and object detection, while RNNs are useful for tasks involving time series data, such as predicting environmental variables over time.

Once the deep learning model has been trained, it can be used to make predictions on new satellite imagery data. This may involve applying the model to a single image or a time series of images, depending on the application.

Overall, deep learning has shown great promise for analyzing satellite imagery data, and is likely to become an increasingly important tool for a wide range of applications in the coming years

Motivation

With the increasing availability of satellite data and the growing importance of artificial intelligence and machine learning in many fields, there is a high demand for individuals with expertise in satellite imagery using deep learning. We look forward to gain knowledge and skills in this area may have a competitive advantage in the job market.

Roles and Contribution to Our Project

Naga Sri Vandana Parimala Sure

Role: Data Processing and model training

Contribution: I have worked on cleaning and pre-processing the Dubai dataset(Kaggle) with segmentation images, such as landmarks, and engineering new features such as radiometric calibration and ortho-rectification. I have also conducted exploratory data analysis to identify trends and patterns in the data.

I have done researched different models suitable for several imagery applications, selecting appropriate models (CNN, SOMS (Self-Organizing Maps), and Auto-encoders.

Venkata Nikith Pulgam

Role: Model deployment and Evaluation

Contribution: I have developed and trained the CNN and SOM models,Rsvd, tuning hyperparameters and evaluating their performance using metrics such as Jaccard Index and Silhouette Score. I have also conducted comparative analysis of the different models and presented the results in the project report.

For Visualization and reporting: We both have involved in creating visualizations of the model outputs such as Netron (3rd party app) and preparing a report summarizing the findings of the project. Overall , we both shared our ideas and helped each other by doing all of these tasks

Related Work

Paper 1:

https://www.sciencedirect.com/science/article/abs/pii/S0378383922000229

Multispectral satellite imagery and machine learning for the extraction of shoreline indicators

The analysis of shoreline change is crucial for coastal scientists, engineers, and managers, with Multispectral Satellite Imagery (MSI) providing high-resolution datasets that allow for frequent monitoring on a global scale. The availability of free Landsat and Sentinel-2 MSI datasets has increased the frequency of studies on coastal change. However, current methods for shoreline extraction from MSI are limited as they do not capture all shoreline types and indicators beyond the waterline. To address this, a review paper presents various techniques for shoreline feature extraction, including water indexing, Machine Learning (ML), and segmentation methods. The paper provides a comprehensive review of the current methods for shoreline extraction from MSI, highlighting the gaps and challenges in using this approach to understand global

shoreline changes. This roadmap can help guide future efforts to improve the use of MSI for understanding shoreline changes.

Paper 2: https://www.mdpi.com/2072-4292/11/15/1762

Oil Spill Identification from Satellite Images Using Deep Neural Networks

Oil spills pose a major threat to marine and coastal environments, making it crucial to detect them quickly and accurately. Synthetic aperture radar (SAR) sensors are commonly used for this purpose, due to their ability to operate in various weather and illumination conditions. However, discriminating between oil spills and other dark spots can be challenging. To address this, researchers have proposed using deep convolutional neural networks (DCNNs) for semantic segmentation, and have introduced a publicly available SAR image dataset to serve as a benchmark for future oil spill detection methods. DeepLabv3+ showed the best performance in terms of test set accuracy and inference time. Results suggest that DCNN segmentation models trained and evaluated on the dataset can be used to create efficient oil spill detectors. This work is expected to make significant contributions to future research in oil spill identification and SAR image processing.

Differences to our project:

The first paper focuses on multispectral satellite Imagery at which some of the shoreline extractions are limited but in our project, all the information is retained as we have divided the images into tiles and masks (along with labels) to classify satellite images. Each image is transformed into 256 x256x3 (here it gets converted into one hot encoding) which helps the images to adjust the corresponding axis.

The second paper focuses on oil spill detection where DCCN, SAR image dataset is used. Deeplabv3+ model is used to improve the testing accuracy although it is not valid for all datasets in the case of our project, we were beneficial with the segmentation of the Dubai dataset where we can work with any case regarding the significant contribution towards our applications.

Dataset

The dataset comprises 72 aerial images of Dubai captured by MBRSC satellites and labeled with pixel-wise semantic segmentation for six distinct classes. The images are grouped into six larger tiles. The six classes in the dataset are as follows:

1. **Building**: labeled with the color code #3C1098
2. **Land (unpaved area**): labeled with the color code #8429F6
3. **Road**: labeled with the color code #6EC1E4
4. **Vegetation**: labeled with the color code #FEDD3A
5. **Water**: labeled with the color code #E2A929
6. **Unlabeled**: labeled with the color code #9B9B9B

The annotations provide a detailed pixel-wise labeling of the aerial imagery, enabling the use of various computer vision and machine learning techniques to analyze the data. The dataset is valuable for applications such as urban planning, environmental monitoring, and disaster response etc.

The data is divided into 8 tiles in which there is a subdivision of images and masks

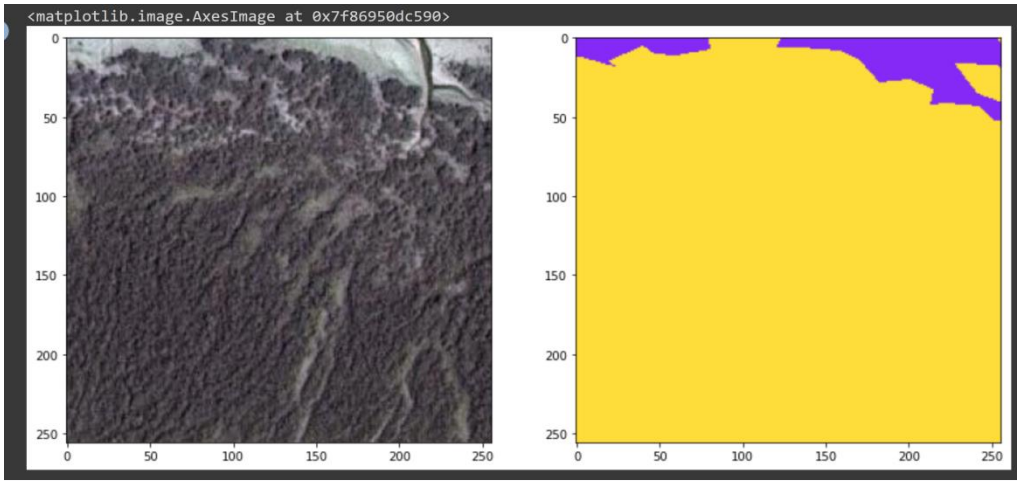
Image: An image is a visual representation of something

Masks: In the context of image processing, masks are binary images that indicate which pixels in an original image should be retained, modified, or discarded during further processing

```
for path, subdirs, files in os.walk(os.path.join(dataset_root_folder, dataset_name)):
  dir_name = path.split(os.path.sep)[-1]
  #print(dir_name)
  if dir_name == 'images':
    images = os.listdir(path)
    print(images)
    for i, image_name in
```

```
['image_part_008.jpg', 'image_part_002.jpg', 'image_part_003.jpg', 'image_part_009.jpg', 'image_part_006.jpg', 'image_part_
['image_part_006.jpg', 'image_part_003.jpg', 'image_part_002.jpg', 'image_part_004.jpg', 'image_part_008.jpg', 'image_part_
['image_part_003.jpg', 'image_part_008.jpg', 'image_part_006.jpg', 'image_part_001.jpg', 'image_part_005.jpg', 'image_part_
['image_part_002.jpg', 'image_part_005.jpg', 'image_part_009.jpg', 'image_part_006.jpg', 'image_part_001.jpg', 'image_part_
['image_part_001.jpg', 'image_part_007.jpg', 'image_part_004.jpg', 'image_part_009.jpg', 'image_part_005.jpg', 'image_part_
['image_part_003.jpg', 'image_part_001.jpg', 'image_part_006.jpg', 'image_part_009.jpg', 'image_part_004.jpg', 'image_part_
['image_part_008.jpg', 'image_part_006.jpg', 'image_part_007.jpg', 'image_part_005.jpg', 'image_part_003.jpg', 'image_part_
['image_part_002.jpg', 'image_part_009.jpg', 'image_part_006.jpg', 'image_part_008.jpg', 'image_part_007.jpg', 'image_part_
```

```
import numpy as np
import matplotlib.pyplot as plt
image = plt.imread('image_part_001.jpg’)
mask = plt.imread(' mask_part_001.jpg')
masked_image = np.copy(image)
masked_image[mask == 0] = 0
plt.imshow(masked_image)
plt.show()
```



Dataset root folder: This folder usually has a descriptive name that identifies the dataset, and may also include metadata or documentation files.

Data–preprocessing steps

One-hot encoding: One-hot encoding is a process of converting categorical data into a numerical representation that can be used in machine learning models.

Category	One-hot encoding
Red	1 0 0
Green	0 1 0
Blue	0 0 1

Feature Extraction: After training a CNN model on satellite imagery, features can be extracted by taking the output of a convolutional layer. The CNN has learned to identify important patterns in the input images, and the extracted features can be used as input to other machine learning models. This allows for downstream tasks such as classification or segmentation to be performed using the high-level representations of the input data.

Reshaping: Reshaping an image involves changing its dimensions or shape while preserving the total number of pixels.

```
reshaped_image = individual_patched_image.reshape(-1, individual_patched_image.shape[-1])

scaler = MinMaxScaler()

scaled_image = scaler.fit_transform(reshaped_image)
```

```
scaled_image =
scaled_image.reshape(individual_patched_image.shape)

scaled_patch = scaled_image[0]
```

fit_transform: fit_transform() is used to standardize the training data X_train, while the same scaling parameters are used to standardize the test data X_test. By doing this, we ensure that the same scaling is applied consistently to both sets, which can help prevent overfitting and improve generalization performance.

```
individual_patched_image
=minmaxscaler.fit_transform(individual_patched_image.reshape(-1,
individual_patched_image.shape[-1]))
```

Quantitative Analysis

Image Normalization is a process of adjusting the range of pixel intensity values in an image to make it more familiar or normal to human perception. This process is commonly used to enhance contrast for better feature extraction or image segmentation. Image Normalization can also help to remove noise from the image, particularly high-frequency and low-amplitude noise.

By normalizing an image, we bring its intensity values into a range that is easy for our visual senses to process. This reduces the strain on our eyes when viewing unclear images and improves our ability to understand what is happening in the image. When an image has poor contrast, we can use normalization to adjust the intensity values so that they are more normal to our senses.

In a normalized image, the mean intensity value is set to zero and the variance of the intensity values is set to one. This helps to ensure that

the image is scaled appropriately for analysis or processing. There are various statistical measures commonly used in data analysis, such as mean, median, standard deviation, minimum and maximum values, among others. These measures provide important insights into the central tendency, variability, and range of the data, and can help us to understand and interpret patterns and trends in the data.

Qualitative analysis:

Multispectral imaging captures image data within specific wavelength ranges across the electromagnetic spectrum.Factors such as small number of spectral bands(3 to 15mm) can be accessed with this type of multispectral imaging. This reflection could distort the accurate measurement of the inherent radiation emitted by the objects.

Methodology

At first we have imported all the necessary libraries necessary for data preparation. Minmax scaler is used to scale the data to either min or max in our code.We have used the Reed-Xiaoli (RX) method. We conducted a comprehensive performance analysis of these methods on scenes with diverse backgrounds and a range of representative targets. The comparative results demonstrate that certain detectors outperform others in specific scenes.

Hyperparameters: Hyperparameters such as epochs, pachify, input size, hidden size, label, and batch size were defined.

Model: Convolutional Neural Network (CNN) is a type of deep learning network architecture that is widely used for image recognition and other pixel data processing tasks. The unique feature of CNNs is the built-in convolutional layer, which reduces the dimensionality of images while retaining essential information.

CNNs are designed with specific layers, including the convolutional layer, ReLU layer, pooling layer, and fully connected layer. These layers work together to extract features from images and classify them. The input image is passed through these layers to produce an output classification.

Overall, CNNs are highly effective in image recognition tasks due to their ability to learn and extract complex features from images, leading to improved accuracy in classification. In a Convolutional Neural Network (CNN), the convolutional layer plays a crucial role in feature extraction by scanning the image using various filters. The output from this layer is a set of feature maps that highlight important features in the input image.

The ReLU layer, which follows the convolutional layer, is responsible for introducing non-linearity to the model by rectifying all negative values to zero. This helps to prevent the network from becoming too linear and enhances its ability to learn complex features.

To increase computational speed and reduce the dimensionality of the feature maps, pooling layers are commonly applied in CNNs. The most widely used pooling technique is max pooling, which computes the maximum value of each subregion.

Finally, the fully connected layer aggregates and weights all the information extracted by the previous layers and generates the final classification output. This layer is responsible for making the ultimate decision about which class the input image belongs to.

We have defined the empty list of the image dataset along with mask dataset to read the image files we need to define the Opencv module along with .imshow() method. There are 8 image files. So further we need to use the for loop to go across all the image files to explore the important features whilst making all the important information enact.

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).

This line depicts the image to convert the image into RGB(3) scale from grey scale.The size and shape of the image is scaled by the modulus division and then reshaping it.

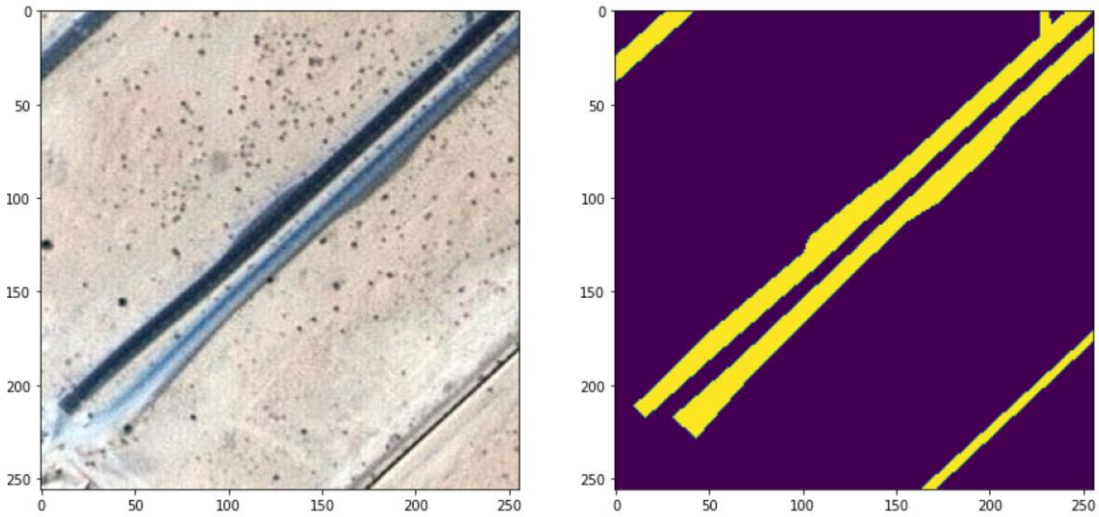
The images are plotted using the matplotlib functions(plt). As with go further into the code ,the features have to be mapped with the one-hot encoding along with labels

```
class_colors = {
    'building': '#3C1098',
```

```
    'land': '#8429F6',
    'road': '#6EC1E4',
    'vegetation': '#FEDD3A',
    'water': '#E2A929',
    'unlabeled': '#9B9B9B'
}
class_rgb = {}
for key, value in class_colors.items():

    rgb = [int(value[i:i+2], 16) for i in (0, 2, 4)]
    class_rgb[key] = np.array(rgb)
    for key, value in class_rgb.items():
    print(key + ': ' + str(value));
```

Similarly the mask images are defined as shown below:

```
<matplotlib.image.AxesImage at 0x7f8693390e10>
```



To convert the labels into categorical

from tensorflow.keras.utils import to_categorical

labels_categorical_dataset = to_categorical(labels, num_classes=total_classes)

Now the most important part inserting all the data into cnn model,som model and then from there to auto-encoders along with several randomized Singular value decomposition. rSVD is an algorithm used for dimensionality reduction of large matrices. It approximates the original matrix by randomly projecting it onto a lower-dimensional space and then factorizing the result into smaller matrices. The process is repeated iteratively until the desired level of accuracy is achieved.

from scipy.sparse.linalg import svds

import numpy as np

features, labels = 1000, 500

data_matrix = np.random.rand(features, labels)

k = 10(target rank)

U, S, Vt = svds(data_matrix, k=k)(computing svds)

Here u s,vt are singular vectors and singular values to obtain the final approximation.

data_matrix_approx = U.dot(np.diag(S)).dot(Vt)

(reconstruction of low ranked matrix)

Convolution model

In the context of image noise reduction or coloring, CNNs are often utilized as part of an Autoencoder architecture, where they are employed to encode and decode image data

```
def multi_unet_model(n_classes=5, image_height=256, image_width=256, image_channels=1):

  inputs = Input((image_height, image_width, image_channels))

  source_input = inputs

  c1 = Conv2D(16, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(source_input)
  c1 = Dropout(0.2)(c1)
  c1 = Conv2D(16, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c1)
  p1 = MaxPooling2D((2,2))(c1)

  c2 = Conv2D(32, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(p1)
  c2 = Dropout(0.2)(c2)
  c2 = Conv2D(32, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c2)
  p2 = MaxPooling2D((2,2))(c2)

  c3 = Conv2D(64, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(p2)
  c3 = Dropout(0.2)(c3)
  c3 = Conv2D(64, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c3)
  p3 = MaxPooling2D((2,2))(c3)
```

```
c4 = Conv2D(128, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(p3)
c4 = Dropout(0.2)(c4)
c4 = Conv2D(128, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c4)
p4 = MaxPooling2D((2,2))(c4)

c5 = Conv2D(256, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(p4)
c5 = Dropout(0.2)(c5)
c5 = Conv2D(256, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c5)

u6 = Conv2DTranspose(128, (2,2), strides=(2,2), padding="same")(c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(128, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(u6)
c6 = Dropout(0.2)(c6)
c6 = Conv2D(128, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c6)

u7 = Conv2DTranspose(64, (2,2), strides=(2,2), padding="same")(c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(64, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(u7)
c7 = Dropout(0.2)(c7)
c7 = Conv2D(64, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c7)

u8 = Conv2DTranspose(32, (2,2), strides=(2,2), padding="same")(c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(u8)
c8 = Dropout(0.2)(c8)
c8 = Conv2D(32, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c8)

u9 = Conv2DTranspose(16, (2,2), strides=(2,2), padding="same")(c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(u9)
c9 = Dropout(0.2)(c9)
c9 = Conv2D(16, (3,3), activation="relu", kernel_initializer="he_normal", padding="same")(c9)
```

```
outputs = Conv2D(n_classes, (1,1), activation="softmax")(c9)

model = Model(inputs=[inputs], outputs=[outputs])
return model
```

```
Total params: 1,941,190
Trainable params: 1,941,190
Non-trainable params: 0
```

Dice Coefficient

In the context of image segmentation tasks, the Dice coefficient is a commonly used metric for evaluating the performance of a segmentation model. The Dice coefficient measures the overlap between the predicted and ground truth segmentation masks and ranges from 0 to 1, where a value of 1 indicates perfect overlap and 0 indicates no overlap at all.

The Dice loss is then defined as 1 minus the Dice coefficient, which means that the goal of the model is to minimize this loss function.

However, in some cases, the segmentation model may struggle with rare or difficult-to-segment classes. This is where the Focal Loss comes in. The Focal Loss is designed to give more weight to difficult examples, by downweighting easy examples that are already well classified. This helps the model to focus more on the difficult classes and improve its performance overall.

To balance the two losses, we can combine them using a weighted sum, where we give more weight to the Dice loss (since it is the primary loss function) and a smaller weight to the Focal Loss. For example, we could set the weight of the Focal Loss to 1, so the Total Loss becomes the sum of the Dice loss and the Focal Loss.

Therefore, the formula for the Total Loss can be written as:

Total Loss = (Dice loss + (1 * Focal Loss))

This way, the model tries to optimize both the Dice coefficient and the Focal Loss simultaneously, leading to better segmentation performance.

JACCARD INDEX

The hyper-parameters used in our project are intersection,union and iou

```
def jaccard_index('image_part_001, 'image_part_002):
    intersection = np.logical_and('image_part_001, 'image_part_002)
```

```
union = np.logical_or('image_part_001, 'image_part_002)
iou = np.sum(intersection) / np.sum(union)
return iou
```

Convolution: The hyper-parameters used here is c1,c2,c3,c4,c5,c6,c7,c8,c9 where (3,3)matrix is used along with the variation of 256 to 32.The other parameters are kernel initializers, padding, activation functions etc. The dropout layer is used to remove the overfitting in the model and max-pooling is used to return the maximum value in the convolution.

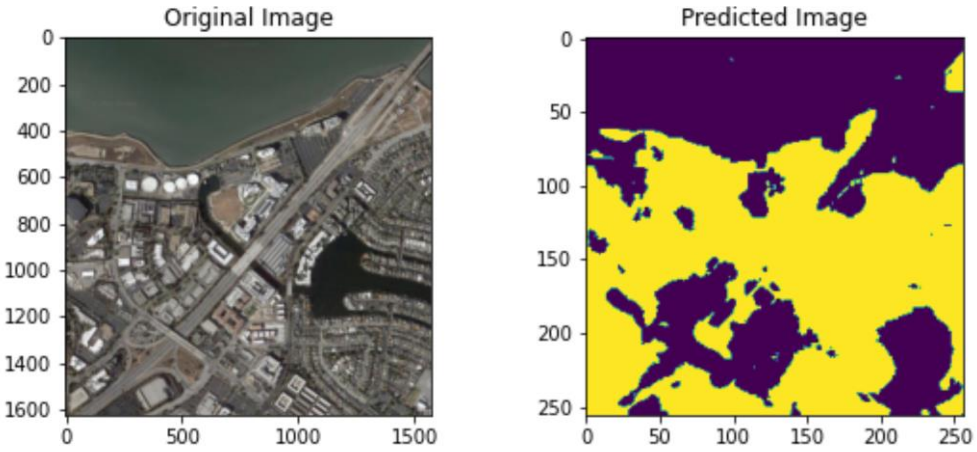
U-net architecture: The hyper-parameters we used here is Number of Convolutional Blocks, Number of Filters, Dropout Probability, Learning Rate, Batch Size, Input Size and Activation Function.

Batch Size: The batch size determines the number of samples that are processed in each training iteration. A larger batch size can result in faster training but can also require more memory.

Dropout Probability: Dropout is a regularization technique used to prevent overfitting. The dropout probability determines the percentage of neurons that are randomly dropped out during training.

Experiments and Results:

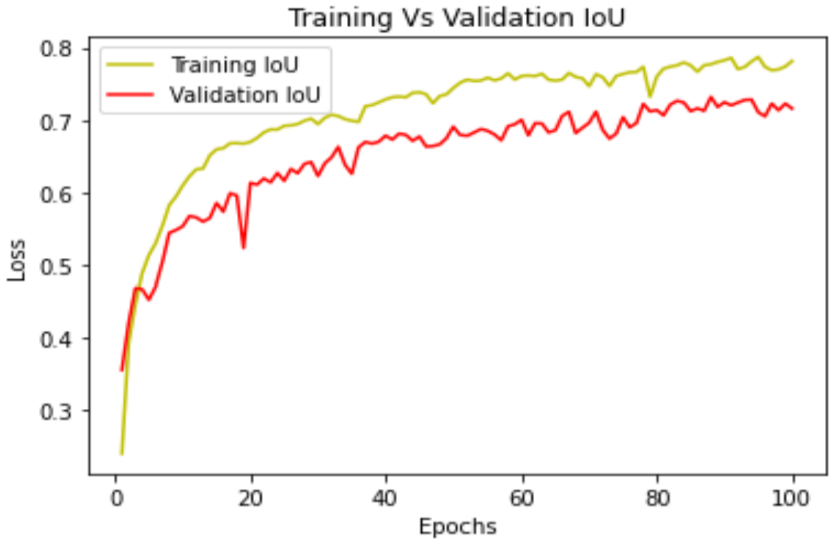
The below image shows the overall image segmentation and classification of original image and the predicted image



To evaluate the performance of the model using jaccard coefficient

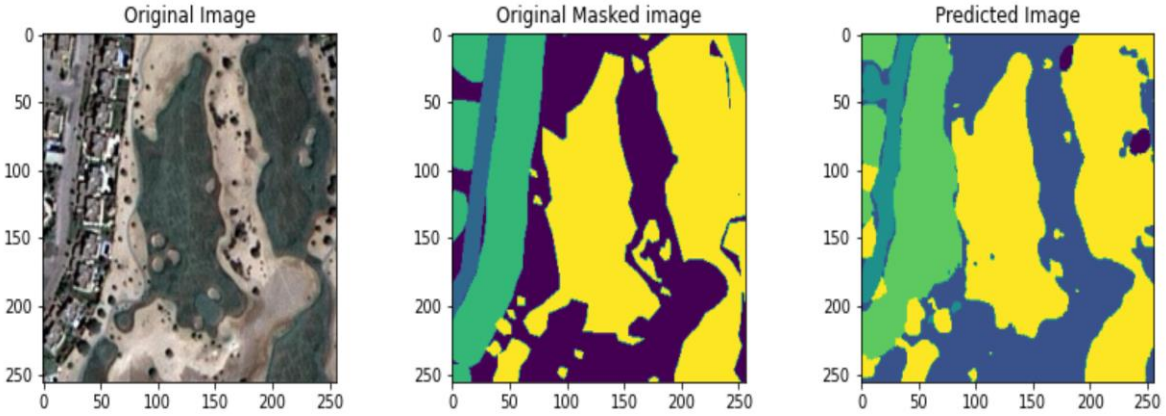
```
jaccard_coef = history_a.history['jaccard_coef']
val_jaccard_coef = history_a.history['val_jaccard_coef']

epochs = range(1, len(jaccard_coef) + 1)
plt.plot(epochs, jaccard_coef, 'y', label="Training IoU")
plt.plot(epochs, val_jaccard_coef, 'r', label="Validation IoU")
plt.title("Training Vs Validation IoU")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

To plot the graph of training set and validation set using Similarity coefficient.



We have achieved the accuracy of 89% and reduced the loss upto 88%

```
51/51 [==============================] - 9s 185ms/step - loss: 0.8884 - accuracy: 0.8972 - jaccard_coef: 0.7814 - val_loss: 0.9190 - val_accuracy: 0.8
492 - val_jaccard_coef: 0.7158
```



Conclusion:

In summary, deep learning-based satellite imagery detection has proven to be a powerful tool for analyzing and categorizing large volumes of satellite imagery data. It has a broad range of potential applications, including disaster response and management, urban planning, environmental monitoring, and military intelligence. The use of deep learning allows for faster and more accurate analysis of data, providing a more complete understanding of the world around us.

However, there are some challenges that must be addressed in the use of deep learning for satellite imagery detection, such as the need for large amounts of high-quality training data, difficulty in detecting small or obscured objects, and the potential for algorithmic bias.

Future Work:

Looking to the future, deep learning-based satellite imagery identification has enormous potential in various sectors. It can be used for environmental monitoring, agriculture, urban planning, transportation, defense, and security. With the advancement of deep learning techniques and the availability of high-resolution satellite imagery, we can expect to see even more significant advances in this field in the years to come.

Overall, the future scope of satellite imagery identification using deep learning is vast, promising, and can revolutionize several industries, leading to a better understanding of our planet and enabling better decision-making.

THANK YOU