REVIEW 1 - FINAL YEAR PROJECT (2021 - 2022)

Building extraction and classification of aerial images of rooftop for estimating maximal PV panel installation

Team Members

Shruthi M - 2018103592

Gayathri M - 2018103535

Jayapriya M - 2018103029

Under the guidance of

Prof Dr. P. Uma Maheshwari

PROBLEM STATEMENT

Sustainable environment is required for the advancement of economic development,to improve energy security, access to energy, and mitigate climate change. Energy production sources such as coal, oil, and natural gas are responsible for one-third of global greenhouse gas emissions, and there is a growing need for everyone to switch to solar power for electricity generation. Estimating a roof's solar potential, on the other hand, is a time-consuming process that requires manual labor and site inspection. The current algorithms only work with LIDAR data and do not predict the number of solar PV panels based on the type of rooftops. Furthermore, mapping urban buildings for rooftop segmentation presents its own set of issues, since aerial satellite photos are typically of low resolution. In this project, we use the AIRS dataset that provides a wide coverage of aerial imagery with 7.5 cm resolution and propose a mechanism to address the above problem: (i) Using state-of-the-art MultiRes U-Net architecture for building detection as a first step to identify and segment buildings from aerial image, (ii) Classifying rooftops from the extracted buildings with different deep learning and transfer learning algorithms and (iii) Use maximum fitting algorithm to find the maximal no of solar PV panels based on the type of roof. This proposed solution uses a single drone/satellite image to recognize and classify rooftops for buildings dispersed throughout an area, automating the entire process and reducing human labor.

INTRODUCTION

Over the last few decades, climate change has become a global problem, affecting a variety of life forms as well as the ecosystem as a whole. There is a rising awareness that harnessing renewable resources is the way to go in order to construct a sustainable environment. Potential tapping of solar power for generating electricity has gained enormous popularity and attention in recent years, and people are increasingly gravitating toward the PV revolution. However, traditional approaches, such as online assessment are time-consuming and expensive, and they require a significant amount of human effort, as concerned individuals must visit the site to inspect the building in order to determine whether PV panels should be installed. By automating the process of building roof extraction for PV panel placement, a lot of money and time can be saved. PV sales can be made more efficient by using an automatic PV system design.

We propose a 3-step mechanism as a solution to address this. We use the AIRS dataset that provides a wide coverage of aerial imagery with 7.5 cm resolution. The training dataset contains 857 images and corresponding roof labels with 94 images in validation and 96 images in testing set. The first stage is building detection from aerial satellite images. Image segmentation has gained huge traction with applications in sectors as diverse as medicine, pathology, and geo-sensing. Building segmentation, which is widely utilized in urban planning, topography mapping, disaster assessment, analyzing geographical land occupation, and other applications, has become a recent subject of interest due to the abundance of high-resolution remote sensing data. Despite this, low detection precision on aerial pictures limits automatic mapping of buildings. A large portion of the effort entails manually demarcating buildings from satellite imagery. Feature extraction is the traditional method that is used to identify and segment different classes in remote sensing data. Buildings created for various purposes have diverse patterns of roof surface and border, which necessitates the use of state of the art deep learning models to create an exceptionally high-dimensional feature space. The most famous UNet architecture, a CNN based model, that was originally developed for segmenting biomedical images has been used by the authors of [1] for aerial image segmentation with few modifications. Over the years, numerous other models have

been tried, with U-Net model being the baseline one. This includes, but is not limited to, FCN, DeepLabv3, ICTNet. In [6], a novel framework called ICTNet, leverages border localization for classification and reconstruction of buildings. Here, the model combines the localization accuracy and use of context of the UNet network architecture, with Dense and Squeeze-Excitation (SE) layers. Similarly the authors of [2] have used a shallow CNN model with post-processing techniques for building roof segmentation of rooftops in India. However, most of the models weren't accurate in delineating the edges/boundaries of buildings accurately. In this project, we propose a deep learning framework called MultiRes UNet, that has demonstrated efficacy in multimodal bio-medical image segmentation. Instead of using the direct connection as in UNet, an additional path called the ResPath with convolution operations is used to connect the encoder and decoder.

Following building segmentation, we undertake background subtraction of the masked picture from the original aerial image to extract the rooftops and classify them in the second step. This process yields us the rooftops and we intend to manually annotate the rooftops into different classes as mentioned in [5]. The authors of [5] have used shallow CNN model and also pre-trained models like VGG16, ResNet50, EfficientNetB4 for rooftop classification for different types which has yielded an accuracy around 80%. In [8], machine learning models like Support Vector Machine (SVM) have been used for classifying rooftops from LiDAR data in Geneva, Switzerland to estimate solar energy potential with an accuracy of 66%. The study helped us understand how solar roof-shape classification may be used in new building design, retrofitting existing roofs, and efficient solar integration on building rooftops. In a similar way, we plan to apply customized CNN models with pre-trained models such as VGG16, AlexNet, ResNet50 and provide a comparison on the performances of these models. A comparison on the network performance before and after data augmentation can be carried out to understand the significance of data.

Finally, based on the type of roofs, we determine the maximum number of PV panels. A maximum fitting approach is applied here. In case of flat roofs, we get the solar tilt angle and row separation value and perform sliding tiling of PV panels in landscape mode. With tilted roofs, the same operation is performed in both landscape and portrait mode, with the most efficient mode being used.

RELATED WORK

Over the years, image databases and computer vision have gotten a lot of attention. Many breakthroughs in image segmentation have been made as a result of this. Bio-medical and remote sensing are the two industries where segmentation is critical in understanding the images.

With respect to this, the authors of [1] have performed segmentation using a multi-scale convolutional neural network that adopts an encoder-decoder U-Net architecture. Here, a U-Net is constructed as the main network, and the bottom convolution layer of U-Net is replaced by a set of cascaded dilated convolution with different dilation rates and an auxiliary loss function added after the cascaded dilated convolution. The proposed method has achieved an IOU of 74.24 % on the whole dataset that covers regions like Austin, Chicago, Kitsap Co, West Tyrol, Vienna. The U-Net model with auxiliary loss function proposed here has aided in network convergence, and a major advantage is that it does not involve manual features and does not involve preprocessing or post-processing steps. However, the segmentation of middle parts in buildings are misaligned and the bulges on the boundaries are lost in [1]. Yet another major issue is that the algorithm performs well only in one subset (countryside and forest) but not in another.

By scraping data from Google Maps, Vladimir Golovko (et al., 2017) developed a CNN-based solar PV panel recognition system. Pre-processing techniques such as image scaling and sharpening are used, followed by the training of a 6-layer CNN model. Instead of using high resolution color satellite orthoimagery, the authors employed low-quality satellite imagery (Google Maps satellite photos), which allows them to reduce the approach's requirements. Because some solar panels resemble roof tops, poor quality satellite pictures taken from Google Maps have led to erroneous classification and there is no validation on the dataset.

In [3], a mask R-CNN with three steps is proposed to extract buildings in the city of Christchurch from aerial images post-earthquake to recognise small detached residences to understand the havoc caused by it. Feature extraction with ResNet is the first step. The initial step is to extract features using ResNet. The RPN (Regional Proposal Network) is then utilized to locate RoI and filter out the irrelevant bounding boxes (BB) using object and background classification, as well

as BB regression. The background and buildings are then identified by object classification. The RoIAlign method used instead of the RoI Pool gives better feature extraction. Due to a small training dataset, the model was unable to successfully demarcate building edges, resulting in low accuracy and precision when compared to other SOTA models.

[4] uses a combination of image processing techniques, including Adaptive Edge Detection and contours, to segment out rooftop boundaries and obstacles present inside them along with polygon shape approximation. It provides a comparative analysis of the solar potential of buildings. Several types of the rooftop are considered to learn the intra-class variations. Because Google Maps India's satellite resolution is so low, the edges aren't fully identified, and there are outliers plotting solar panels outside of the building's rooftop area.

In [6], the authors have addressed the problem of semantic segmentation of buildings from remote sensor imagery. ICTNet: a novel network with the underlying architecture of a fully convolutional network, infused with feature re-calibrated Dense blocks at each layer in [6] is combined with dense blocks, and Squeeze-and-Excitation (SE) blocks. Dense blocks connect every layer to every other layer in a feed-forward fashion. Along with good gradient propagation they also encourage feature reuse and reduce the number of parameters substantially as there is no need to relearn the redundant feature maps which allows the processing of large patch sizes. Reconstruction is done by extruding the extracted boundaries of the buildings and comparative analysis is made between the two. With no 3D information on the buildings, the authors have used the building boundaries as a proxy for the reconstruction process and have got better overall IoU compared to other methods. The main limitation here is that there is no loss function for the reconstruction accuracy. Furthermore, due to the fact that ground truth photos used for training contain mistakes and are manually generated, there is a large variance in per-building IoU. In addition, the reconstruction accuracy is consistently lower than classification accuracy by an average of 4.43% ± 1.65%.

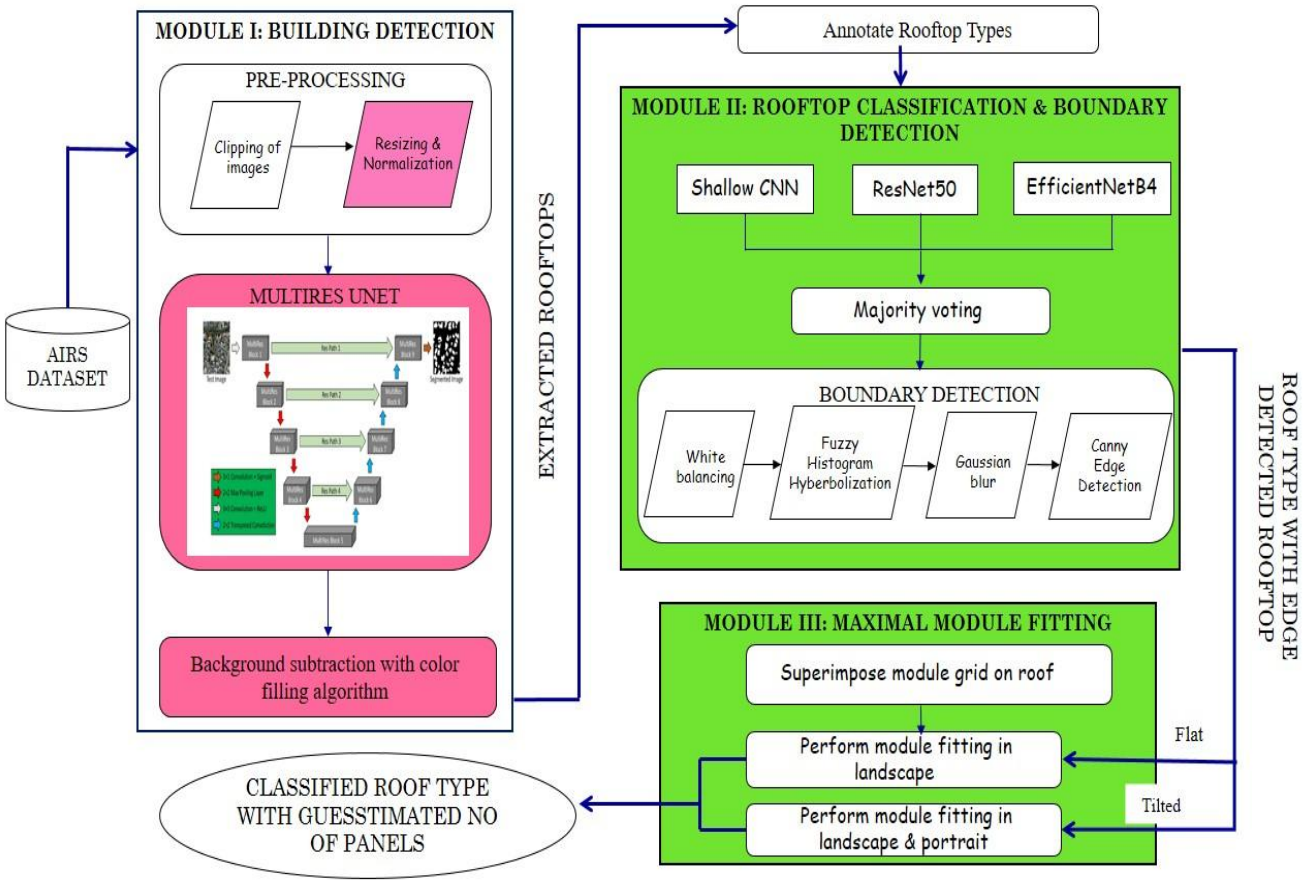
Data preprocessing in [7] involves collecting patch satellite images from Google for the city of Heilbron and manually labeling them. Object proportion distribution in image-level and object occurrence possibility at pixel level is statistically analyzed. SOTA PV segmentation model (DeepSolar) is used to extract visual

features. Local Binary Pattern (LBP) is used for texture feature extraction & color histograms for color feature extraction. The authors have addressed the issue of class imbalance of PV and non-PV panels in rooftops by hard sampling, soft sampling. The major drawback is that lighting conditions caused distinct color clustering groups in PV/Non-PV color clustering, resulting in misclassification along with IOU being less than the acceptable range (0.5) for 1.2m resolution images.

Rooftop classification is the important step in identifying the type of PV panels that can be fitted. M. Buyukdemircioglu , R. Can , S. Kocaman have undertaken research to generate a roof type dataset from very high-resolution (10 cm) orthophotos of Cesme, Turkey, and to classify the roof types using a shallow CNN architecture. UltraCam Falcon large-format digital camera is used in [5] to capture orthophotos with 10cm spatial resolution and roofs are manually classified into 6 different labels. Data augmentation is applied and a shallow CNN architecture is trained. The prediction is investigated by comparing with three different pre-trained CNN models, i.e. VGG-16, EfficientNetB4, and ResNet-50. Simple CNN models are hence easier to implement and require nominal hardware specifications. The shallow CNN model has achieved 80% accuracy. As the roof images were clipped automatically from orthophotos, there are few buildings with overlap. Half-hip roofs are not classified properly and the F1 score obtained for them is very low. The authors haven't experimented with alternate hyperparameter tweaking for the shallow CNN architecture, which is a serious flaw.

Similarly in [8], SVM based machine learning approach is used to classify building roofs in relation to their solar potential. The SVM classifier here on an average produces 66% accuracy and is able to classify rooftop types into 6 major classes. The authors have calculated the ratio of useful roof area for each type of roof shape to that of the corresponding building footprint area and the results are close to 1. This indicates that better the segmentation of building, maximum is the solar potential of each of the rooftop areas.

SYSTEM ARCHITECTURE



Overall System Architecture

Our proposed model's overall system architecture is depicted in the diagram above. The system consists of three major components that work together to address the issues in previous works and add contributions to the existing framework.

The AIRS dataset contains satellite images of Christchurch in New Zealand with 7.5cm resolution. To begin, our proposed system includes two pre-processing steps. The pre-processing stages include clipping satellite photos and the accompanying ground truth masks, as well as scaling and normalization. After that, the model is trained using the MultiRes UNet architecture. MultiRes UNet is chosen here as it has provided great results with image segmentation in previous works. Following this, we perform background subtraction by drawing bounding boxes. The first module mainly emphasizes building detection from satellite images.

The extracted rooftops from the previous module are sent to the next stage. We manually label the extracted rooftops into different classes which are then fed to three different models and a comparative analysis is made. Following that, edge detection of rooftops takes place to mark boundaries on rooftops.

The final module resorts to providing a guesstimate of the number of PV panels that can be fitted in the given rooftop. This is accomplished by applying a maximum fitting algorithm where we move the module grids on rooftops and find the position that gives maximum count.

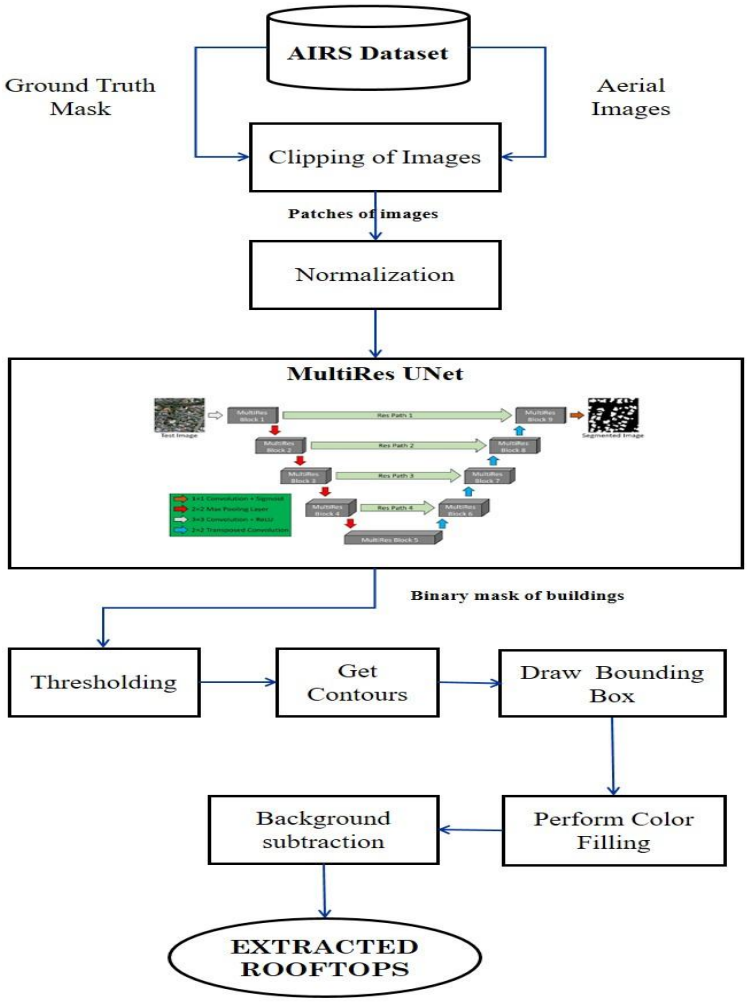
MODULE DESIGN

LIST OF MODULES

MODULE 1: Building Detection.
MODULE II: Roof type classification and boundary detection.
MODULE III: Maximal Fitting algorithm.

MODULE I: BUILDING DETECTION

INPUT: AIRS Dataset
OUTPUT: Rooftops of different buildings



Module Design of Building Detection

- The first step involves **pre-processing** with clipping of large aerial images into smaller tiles and performing resizing and normalization.
  Input: Aerial images.
  Output: Smaller patches of normalized images.

- Following this, the **MultiRes UNet** architecture is implemented.
  Input: Ground truth mask patches and aerial image patches for training.
  Output: Trained model with binary mask for buildings

- The final step involves **background subtraction** performed by drawing bounding boxes around the buildings and color filling them and subtracting the mask from original images.
  Input: Binary mask of buildings.
  Output: Rooftops of different buildings.

PSEUDO-CODE

Pre-Processing

Clipping:

1. Get the original dimensions and the dimensions of smaller patches.
2. Get the stride for cropping images
3. if small_dim % stride != 0
   throw error
4. overlapping := (size / stride) - 1
5. Initialize patches_list := []
6. for i in range (orig_shape / stride - overlapping):
   Crop from i*stride:i*stride+size, j*stride:j*stride+size

Normalization:

1. Resize := cv2.resize(img, (256,256), cv2.INTER_CUBIC).
2. Normalization :=

X_std = (X - X.min(axis=0)) / (X.max - X.min).
X_scaled = X_std * (max - min) + min.

Training the MultiRes UNet Architecture

1. Split dataset into train - 1548 images, validation - 36 images and testing - 144 images.
2. Define the MultiRes model.
3. do:
   3.1 Tweak hyperparameters: Set learning_rate := 0.0001, batch_size := 8, epochs := 100, optimizer := Adam, loss := binary cross entropy
   3.2 Train the model
   3.3 Plot graphs and check on validation data.
  while(find the best model)
4. Save weights for the best model.

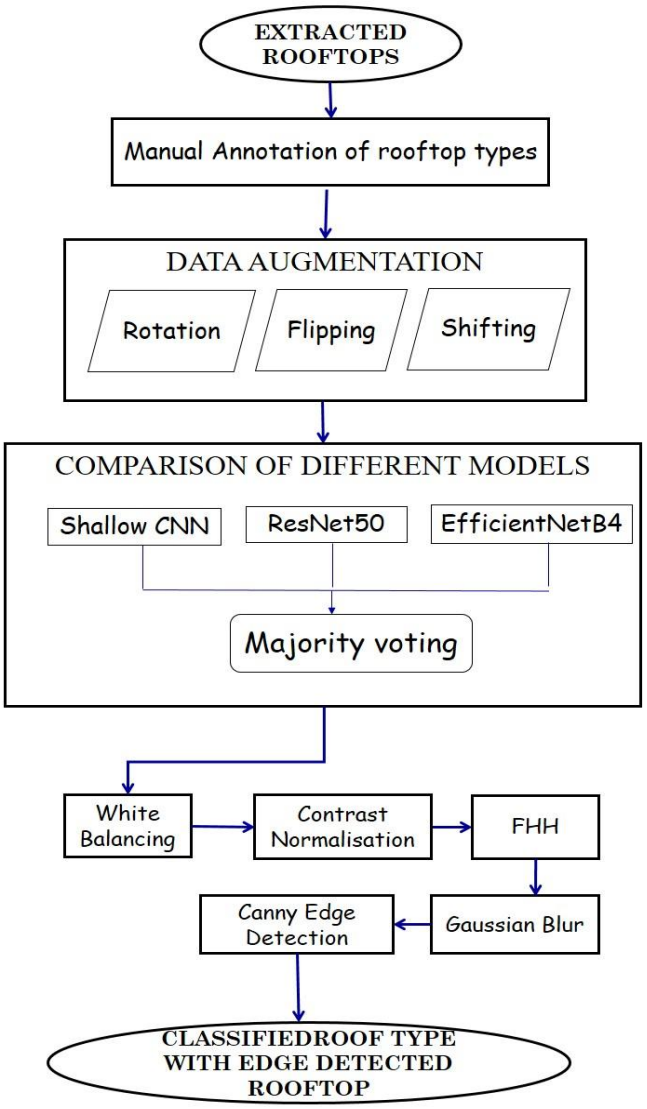
Background Subtraction

Get contours & draw bounding boxes:

1. Convert rgb to grayscale.
2. contours := cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
3. bounding_box_coordinates := (x,y), (x+w,y), (x,y+h), (x+w,y+h)
4. color fill with white color.
5. Remove bg from fg := masked_out_new = np.where(masked_out != 0, masked_out, 255)

MODULE II: ROOF TYPE CLASSIFICATION & BOUNDARY DETECTION

INPUT: Extracted rooftops.
OUTPUT: Classified roof type with edge detected rooftop.



Module Design for Roof type classification & Boundary detection

- The first step involves **manually labeling** the rooftops into 4 different classes - Flat, Gable, Complex, Hip.

- This is followed by **data augmentation** with rotation, flipping and shifting as transformations.

- Following this, we try 3 different models - customized CNN, ResNet50, EfficientNetB4 and perform majority voting
  Input: Rooftops with labels.
  Output: Classified rooftop type.

- The final step involves **boundary detection** performed by contrast normalization, FHH and applying Median and Gaussian blur.
  Input: Classified rooftop type.
  Output: Boundary detected rooftops.

PSEUDO-CODE

Data Augmentation

1. Rotation - Randomly generate an angle (theta) and rotate the image by (theta) degrees clockwise and anticlockwise. ImageDataGenerator(rotation_range = angle theta).
2. Shifting - ImageDataGenerator(width_shift_range = 0.10)
3. Flipping - ImageDataGenerator(horizontal_flip = True)

Comparison of different models

1. Split the dataset into train - 80%, validation - 10% and testing- 10%.
2. Define a customized CNN model
3. Train:
   3.1 Tweak hyperparameters.
   3.2 Plot classification report, confusion matrix
4. Use pre-trained ResNet50 and EfficientNetB4.
5. Fine tune.
6. Repeat step 3 for both the models.
7. For an unseen image:
   7.1 Get predictions from all 3 models.

7.2 Perform majority voting - the class that gets the maximum votes.

Boundary Detection

1. **White patching** - Set a percentile score and remove haze from the image.
white_patch := img_as_ubyte((image*1.0 /
np.percentile(image,percentile, axis=(0, 1))).clip(0, 1))

2. **FHH -** For each pixel, apply the following formula where g_{max}, g_{min} represent the maximum and minimum intensities.

$$\mu(g_{ij}) = \frac{g_{ij} - g_{min}}{g_{max} - g_{min}}$$

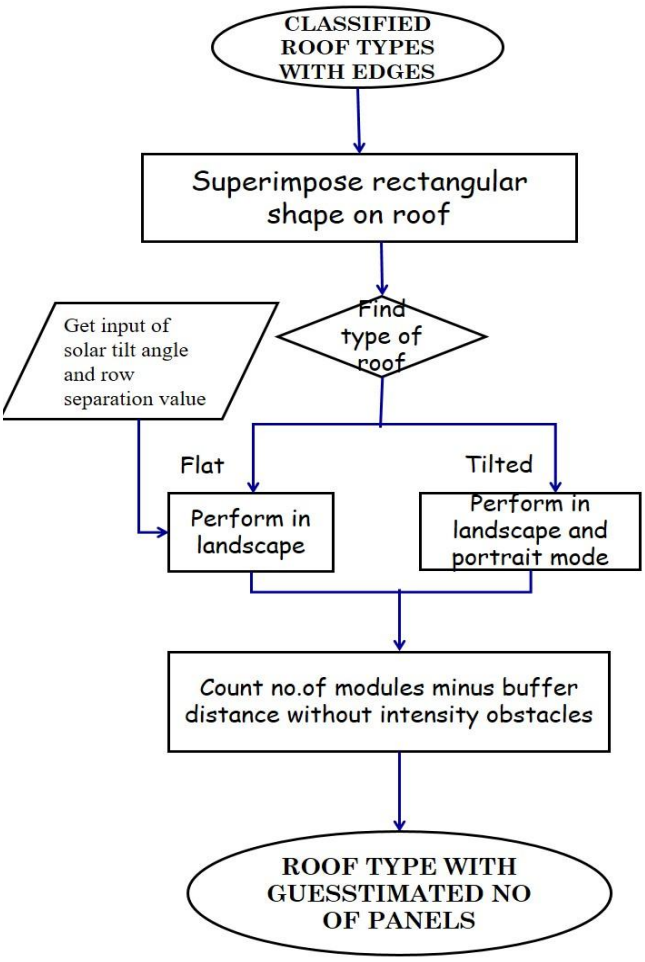
$$\lambda = \frac{L-1}{e^{-1} - 1}$$

3. **Gaussian Blur -** Try for different kernel size and $\boldsymbol{\sigma}$.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

MODULE III: MAXIMAL FITTING ALGORITHM

INPUT: Classified roof type with edge detected rooftop.
OUTPUT: Classified roof type with guesstimated no of panels.



Module Design for Maximal Fitting Algorithm

- ➢ The masked segmented image stored in the new database is taken and the roofs are superimposed with rectangular shape.
- ➢ Then the type of the roof is determined as flat or slope.
    - If the type of the roof is flat, we get user input(solar tilt angle and row separation value) and perform a maximum fitting algorithm on landscape mode.
    - If the type of the roof is slope, we perform a maximum fitting algorithm on landscape and portrait mode.

- ➢ The buffer distance without intensity obstacles is subtracted from the count of the number of modules.
- ➢ The segmented images are fitted with the solar panels.

PSEUDO-CODE

Calculating the PV panels

1. Define the input settings (PV size & tilt) .
2. Create a rectangular module grid according to the settings specified for flat and pitched roofs.
3. Buffer_distance_from_edge = 10cm
3. if flat:
    Shift the module grid with specified row distance in both x and y directions.
   else:
    Shift the module grid with 0 row distance in both x and y directions.
4. For every new position of the grid, count the number of modules within the roof shape minus buffer_distance_from_edge.
5. Find position with most modules being fitted.

IMPLEMENTATION DETAILS

MODULE 1: BUILDING DETECTION

The dataset used here is AIRS dataset that covers almost the full area of Christchurch in New Zealand and provides a wide coverage of aerial imagery with 7.5 cm resolution. The dataset has aerial satellite images along with the corresponding mask images.

(i) Clipping original aerial images

Satellite and aerial images are usually stored as huge images called ‘tiles’ or ‘patches’, which are too large to be segmented directly. The aerial images here are of dimensions 10000 * 10000 pixels and the first step here is to cut large original satellite images into size 1536 * 1536 (as mentioned in base paper). Thus, we obtain 36 smaller patches for a single aerial image.
This is done by sliding window technique where we mention the size of original image (here 10000 * 10000), size of patches (1536 * 1536) and stride length (1536 * 1536 here).

Cut small patches from big satellite images

```
[ ] train_path_image = '/content/drive/MyDrive/airs-minisample/train/image/christchurch_97.tif'
```

- Takes single image and crops the image using sliding window method.
- If stride < size it will do overlapping.

```
def get_patches(img_arr, size, stride):
    if size % stride != 0:
        print("size % stride must be equal 0")

    patches_list = []
    overlapping = 0
    if stride != size:
        overlapping = (size // stride) - 1

    if img_arr.ndim == 3:
        i_max = img_arr.shape[0] // stride - overlapping

        for i in range(i_max):
            for j in range(i_max):
                # print(i*stride, i*stride+size)
                # print(j*stride, j*stride+size)
                patches_list.append(img_arr[i * stride : i * stride + size, j * stride : j * stride + size])
    return np.stack(patches_list)
```

This is done for training, validation, testing images and we get 1548 images in the training set, 72 images in the validation set and 144 images in the testing set.

Result

Original Aerial Image

Patches of image after performing clipping on the original image

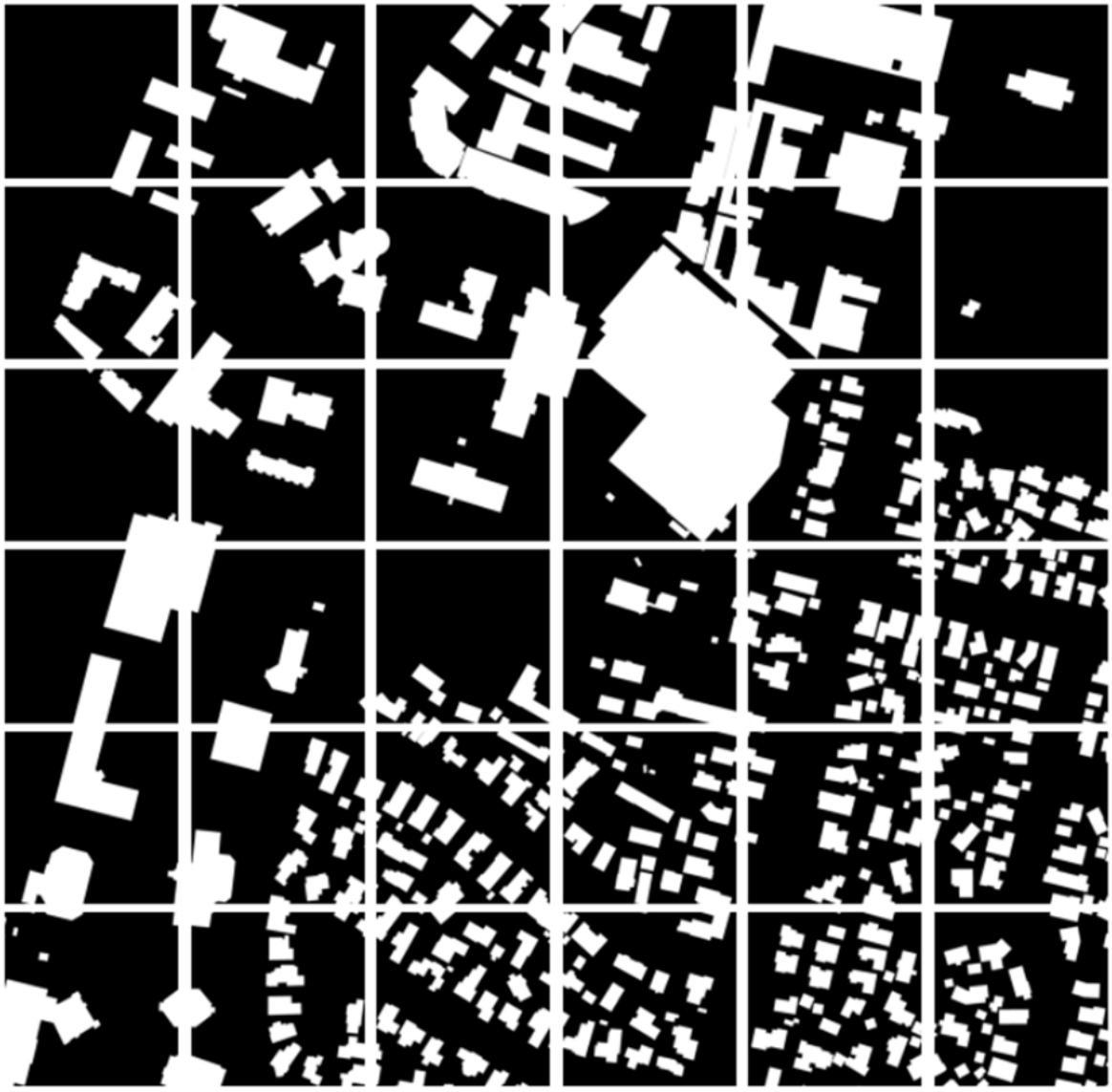
(ii) <u>Clipping corresponding binary mask images</u>

In a similar way, the corresponding mask images are clipped into patches of dimensions 1536 * 1536 and we name the images accordingly.

<u>Result</u>

Groundtruth segmented image

For christchurch_363_vis.tif, x shape: (10000, 10000, 3), x-crops shape: (36, 1536, 1536, 3)



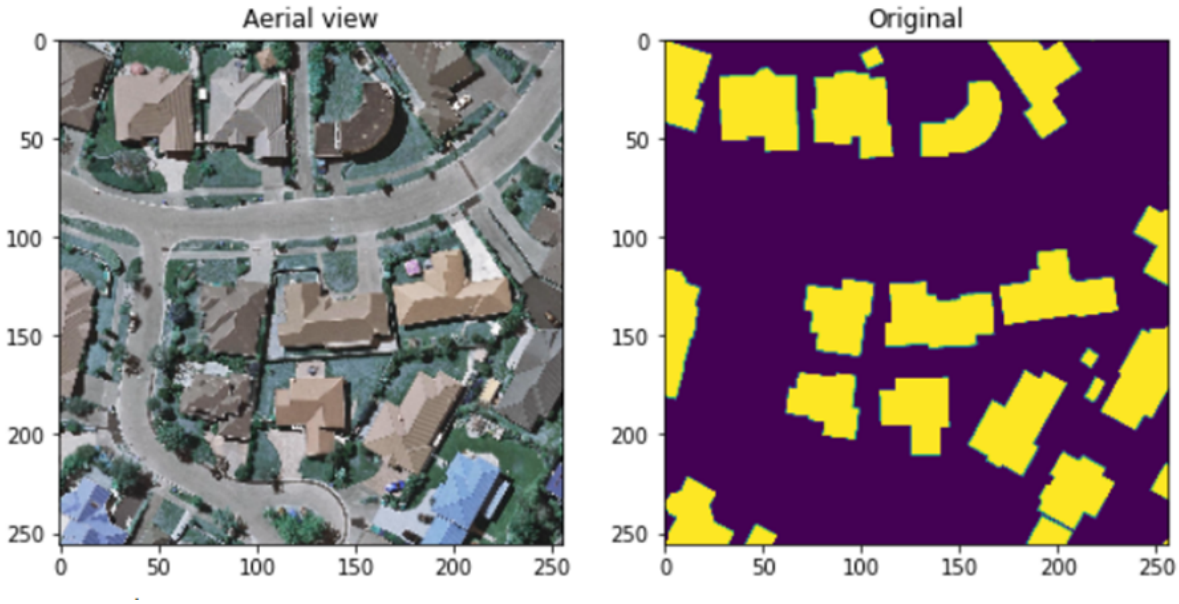
Corresponding patched segmented image after clipping

(iii) <u>Resizing and Normalization</u>

Images of size 1536 * 1536 are still huge to process and train on complex neural nets as we do not have the required hardware specifications. As a result, we resize the images to 256 * 256 with INTER_CUBIC interpolation method as it leads to better resolution of images.

MinMax scaler is employed for normalization as this method is used when the upper and lower boundaries are well known (e.g. for images where pixel intensities go from 0 to 255 in the RGB color range).

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

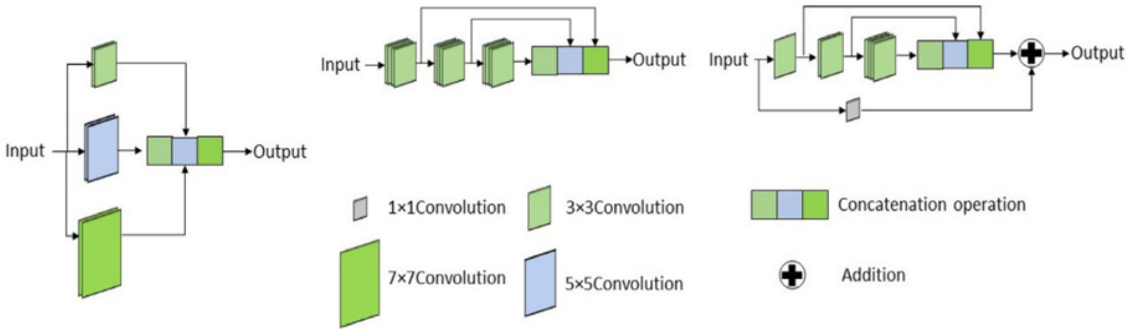


Aerial view and ground truth mask after resizing and employing min-max scaler.

(iv) <u>Adapting the MultiRes UNet architecture</u>

The architecture of the MultiRes UNet network consists of 2 important blocks: the MultiRes Block and the Res Path.

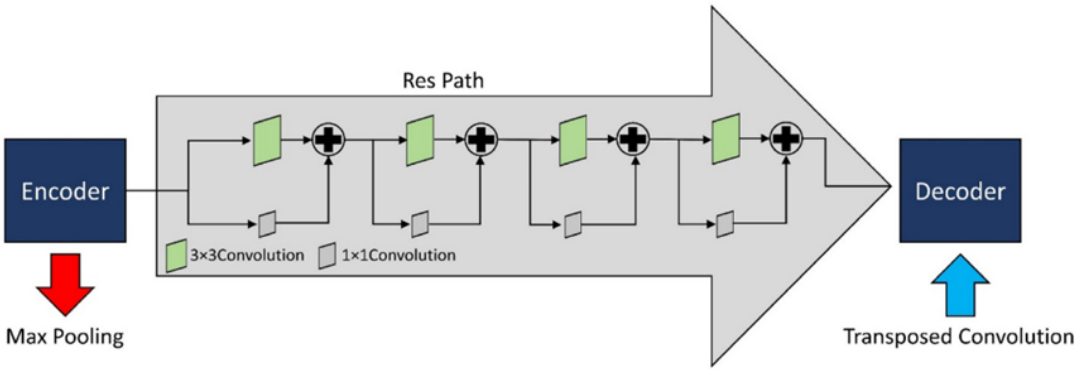
MultiRes Block:
In the U-Net architecture, after each pooling layer and transposed convolutional layer a sequence of two 3 × 3 convolutional layers are used. A series of two 3 × 3 convolutional operations is equivalent to a single 5 × 5 convolutional operation. Therefore, following the approach of Inception network, the simplest way to augment U-Net with multi-resolutional analysis is to incorporate 3 × 3, 5 × 5 and 7 × 7 convolution operations in parallel. Therefore, replacing the convolutional layers with Inception-like blocks should facilitate the U-Net architecture to reconcile the features learnt from the image at different scales. However, the introduction of additional convolutional layers in parallel extravagantly increases the memory requirement. Thus, we factorize the bigger, more demanding 5 × 5 and 7 × 7 convolutional layers, using a sequence of smaller and lightweight 3 × 3 convolutional blocks. The outputs of the 2nd and 3rd 3 × 3 convolutional blocks effectively approximate the 5 × 5 and 7 × 7 convolution operations respectively. We hence take the outputs from the three convolutional blocks and concatenate them together to extract the spatial features from different scales. Though this modification greatly reduces the memory requirement, it is still quite demanding. This is mostly due to the fact that in a deep network if two convolutional layers are present in a succession, then the number of filters in the first one has a quadratic effect over the memory. Therefore, instead of keeping all the three consecutive convolutional layers of equal number of filters, we gradually increase the filters in those (from 1 to 3), to prevent the memory requirement of the earlier layers from exceedingly propagating to the deeper part of the network. We also add a residual connection and introduce a 1 × 1 convolutional layer to comprehend some additional spatial information.



MultiRes Block

Res Path:

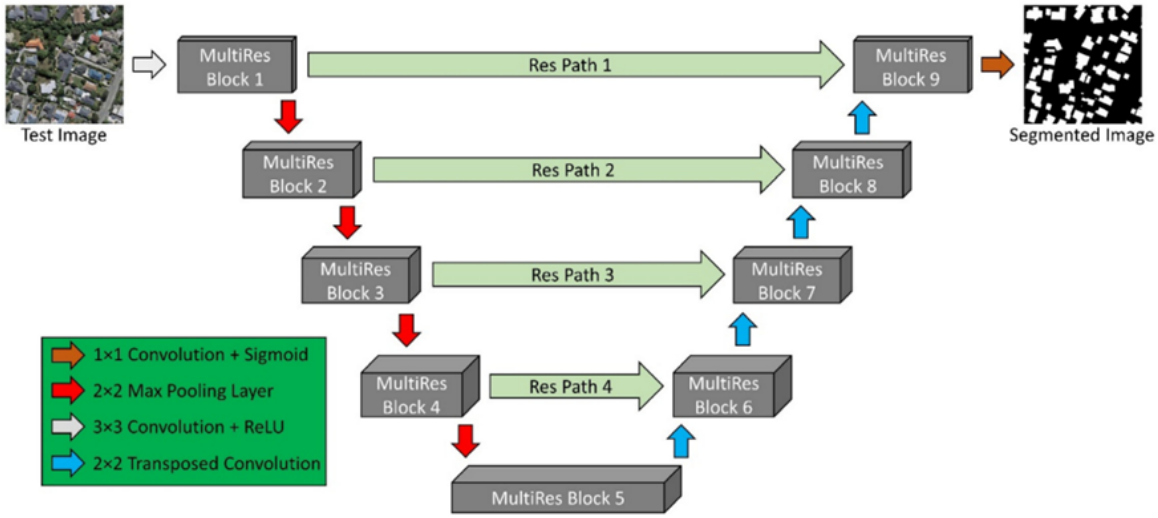
Skip connections enable the network to propagate the spatial information that gets lost during the pooling operation from encoder to decoder. Despite preserving the dissipated spatial features, there could be flaws.The first shortcut connection bridges the encoder before the first pooling with the decoder after the last deconvolution operation. Here, the features coming from the encoder are supposed to be lower level features as they are computed in the earlier layers of the network. On the contrary, the decoder features are supposed to be of much higher level, since they are computed at the very deep layers of the network. Therefore, they go through more processing. Hence, there is a possible semantic gap between the two sets of features being merged. This can be attributed to the fact that, not only the features from the encoder are going through more processing, but also we are fusing them with decoder features of much juvenile layers. Therefore, to alleviate the disparity between the encoder-decoder features, convolutional layers are incorporated along the shortcut connections. Our hypothesis is that these additional non-linear transformations on the features propagating from the encoder stage should account for the further processing done during the by decoder stage therein. Furthermore, instead of using the usual convolutional layers, residual connections are introduced here and they are concatenated with the decoder features.



Res Path

Architecture:

Four MultiRes blocks are each used in the encoder and decoder stage. The number of filters in each of the MultiRes blocks is based on the formula: $W = \alpha \times U$ where α is the scalar coefficient whose value is set to 1.67 and U refers to the no of filters. The parameter W preserves an analogous connection between the suggested MultiRes-UNet network and the main UNet network. After every pooling or transposing of layers, the value of W became double, similar to the original UNet network. The values of U = [32, 64, 128, 256, 512] are set as follows. We also allocated filters of [W/ 6], [W/ 3], and [W/ 2] to the three succeeding convolutions respectively. The filters used in the Res Path are as follows: 64, 128, 256, 512. Batch Normalization is performed in each of the blocks to avoid overfitting. ReLU is used as the activation function in all the convolution layers and the last layer employs the Sigmoid activation function as we have only binary classes here (0 indicating background, 1 indicating the building).



Architecture of MultiRes UNet

Hyper parameters	**Values**
Learning rate	0.0001
Epochs	100
Batch size	8
Image dimensions	256 × 256
Optimizer	Adam
Loss	Binary cross entropy
Activation Function	ReLU (in all convolution layers) Sigmoid (in the output layer)

Result

```
def MultiResBlock(U, inp, alpha = 1.67):

    W = alpha * U

    shortcut = inp

    shortcut = conv2d_bn(shortcut, int(W*0.167) + int(W*0.333) +
                         int(W*0.5), 1, 1, activation=None, padding='same')

    conv3x3 = conv2d_bn(inp, int(W*0.167), 3, 3,
                        activation='relu', padding='same')

    conv5x5 = conv2d_bn(conv3x3, int(W*0.333), 3, 3,
                        activation='relu', padding='same')

    conv7x7 = conv2d_bn(conv5x5, int(W*0.5), 3, 3,
                        activation='relu', padding='same')

    out = concatenate([conv3x3, conv5x5, conv7x7], axis=3)
    out = BatchNormalization(axis=3)(out)

    out = add([shortcut, out])
    out = Activation('relu')(out)
    out = BatchNormalization(axis=3)(out)

    return out
```

MultiRes Block

```
def ResPath(filters, length, inp):

    shortcut = inp
    shortcut = conv2d_bn(shortcut, filters, 1, 1,
                         activation=None, padding='same')

    out = conv2d_bn(inp, filters, 3, 3, activation='relu', padding='same')

    out = add([shortcut, out])
    out = Activation('relu')(out)
    out = BatchNormalization(axis=3)(out)

    for i in range(length-1):

        shortcut = out
        shortcut = conv2d_bn(shortcut, filters, 1, 1,
                             activation=None, padding='same')

        out = conv2d_bn(out, filters, 3, 3, activation='relu', padding='same')

        out = add([shortcut, out])
        out = Activation('relu')(out)
        out = BatchNormalization(axis=3)(out)

    return out
```

Res Path

```
def MultiResUnetBP(height, width, n_channels):

    inputs = Input((height, width, n_channels))

    mresblock1 = MultiResBlock(32, inputs)
    pool1 = MaxPooling2D(pool_size=(2, 2))(mresblock1)
    mresblock1 = ResPath(32*2, 4, mresblock1)

    mresblock2 = MultiResBlock(32*2, pool1)
    pool2 = MaxPooling2D(pool_size=(2, 2))(mresblock2)
    mresblock2 = ResPath(32*4, 3, mresblock2)

    mresblock3 = MultiResBlock(32*4, pool2)
    pool3 = MaxPooling2D(pool_size=(2, 2))(mresblock3)
    mresblock3 = ResPath(32*8, 2, mresblock3)

    mresblock4 = MultiResBlock(32*8, pool3)
    pool4 = MaxPooling2D(pool_size=(2, 2))(mresblock4)
    mresblock4 = ResPath(32*16, 1, mresblock4)

    mresblock5 = MultiResBlock(32*16, pool4)

    up6 = concatenate([Conv2DTranspose(
        32*8, (2, 2), strides=(2, 2), padding='same')(mresblock5), mresblock4], axis=3)
    mresblock6 = MultiResBlock(32*8, up6)

    up7 = concatenate([Conv2DTranspose(
        32*4, (2, 2), strides=(2, 2), padding='same')(mresblock6), mresblock3], axis=3)
    mresblock7 = MultiResBlock(32*4, up7)
```

```
    up8 = concatenate([Conv2DTranspose(
        32*2, (2, 2), strides=(2, 2), padding='same')(mresblock7), mresblock2], axis=3)
    mresblock8 = MultiResBlock(32*2, up8)

    up9 = concatenate([Conv2DTranspose(32, (2, 2), strides=(
        2, 2), padding='same')(mresblock8), mresblock1], axis=3)
    mresblock9 = MultiResBlock(32, up9)

    conv10 = conv2d_bn(mresblock9, 1, 1, 1, activation='sigmoid')

    MultiResModel = Model(inputs=[inputs], outputs=[conv10])

    return MultiResModel
```

Brief Model Summary

```
MultiResModel = MultiResUnetBP(height=256, width=256, n_channels=3)
MultiResModel.summary()
```

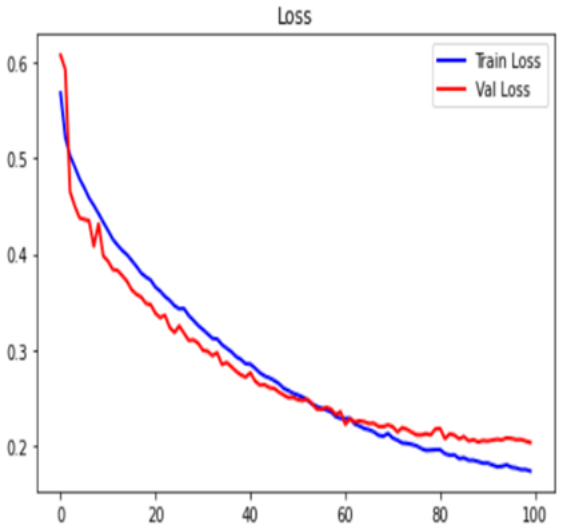
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv2d_1 (Conv2D)              (None, 256, 256, 8)  216         ['input_1[0][0]']

 batch_normalization_1 (BatchNo  (None, 256, 256, 8)  24         ['conv2d_1[0][0]']
 rmalization)

 activation (Activation)        (None, 256, 256, 8)  0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 256, 256, 17  1224        ['activation[0][0]']
                                )

 batch_normalization_2 (BatchNo  (None, 256, 256, 17  51         ['conv2d_2[0][0]']
 rmalization)                   )

 activation_1 (Activation)      (None, 256, 256, 17  0           ['batch_normalization_2[0][0]']
                                )

 conv2d_3 (Conv2D)              (None, 256, 256, 26  3978        ['activation_1[0][0]']
                                )

 batch_normalization_3 (BatchNo  (None, 256, 256, 26  78         ['conv2d_3[0][0]']
 rmalization)                   )

 activation_2 (Activation)      (None, 256, 256, 26  0           ['batch_normalization_3[0][0]']
                                )
```

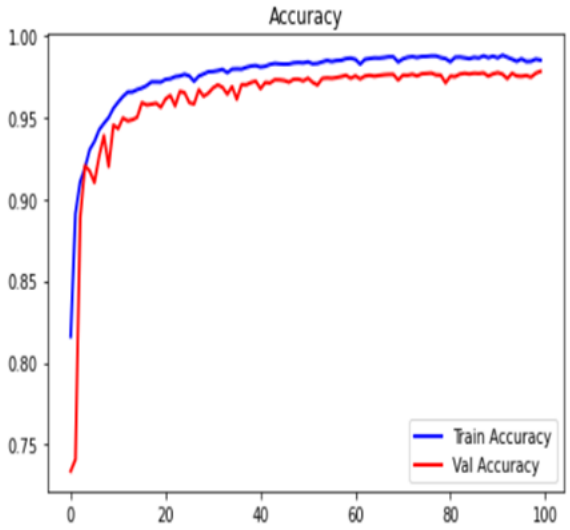
```
 concatenate_12 (Concatenate)   (None, 256, 256, 51  0           ['activation_52[0][0]',
                                )                                  'activation_53[0][0]',
                                                                   'activation_54[0][0]']

 batch_normalization_78 (BatchN  (None, 256, 256, 51  153        ['conv2d_52[0][0]']
 ormalization)                  )

 batch_normalization_82 (BatchN  (None, 256, 256, 51  204        ['concatenate_12[0][0]']
 ormalization)                  )

 add_18 (Add)                   (None, 256, 256, 51  0           ['batch_normalization_78[0][0]',
                                )                                  'batch_normalization_82[0][0]']

 activation_55 (Activation)     (None, 256, 256, 51  0           ['add_18[0][0]']
                                )

 batch_normalization_83 (BatchN  (None, 256, 256, 51  204        ['activation_55[0][0]']
 ormalization)                  )

 conv2d_56 (Conv2D)             (None, 256, 256, 1)  51          ['batch_normalization_83[0][0]']

 batch_normalization_84 (BatchN  (None, 256, 256, 1)  3          ['conv2d_56[0][0]']
 ormalization)

 activation_56 (Activation)     (None, 256, 256, 1)  0           ['batch_normalization_84[0][0]']

==================================================================================================
Total params: 9,906,494
Trainable params: 9,876,980
Non-trainable params: 29,514
__________________________________________________________________________________________________
```

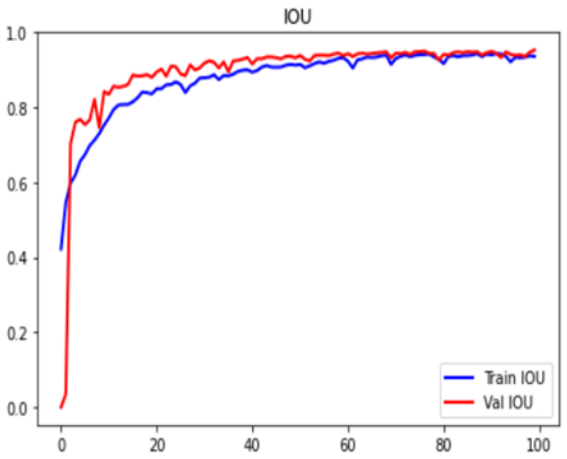
```
[ ] history = MultiResModel.fit(x = train_X_scaler, y = train_Y_scaler, validation_data = (val_X, val_Y), batch_size = 8, epochs = 100, verbose = 1)
```

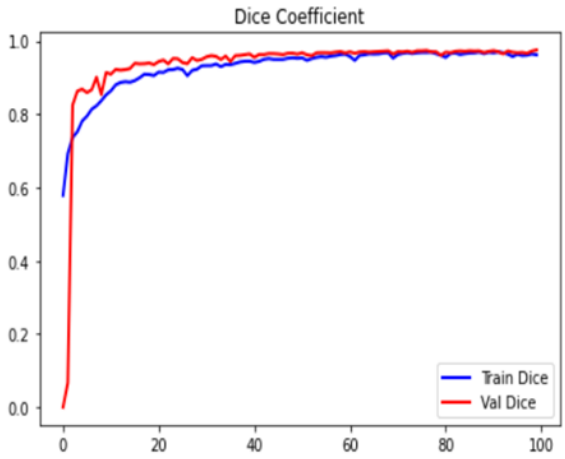
```
Epoch 1/100
194/194 [==============================] - 213s 1s/step - loss: 0.5683 - IOU: 0.4220 - mcc: 0.5200 - dice_coef: 0.5787 - accuracy: 0.8161 - val_loss: 0.6078 - val_IOU: 5.4112e-05 - v
Epoch 2/100
194/194 [==============================] - 198s 1s/step - loss: 0.5220 - IOU: 0.5455 - mcc: 0.6563 - dice_coef: 0.6931 - accuracy: 0.8912 - val_loss: 0.5929 - val_IOU: 0.0348 - val_m
Epoch 3/100
194/194 [==============================] - 198s 1s/step - loss: 0.5028 - IOU: 0.5982 - mcc: 0.7049 - dice_coef: 0.7369 - accuracy: 0.9108 - val_loss: 0.4651 - val_IOU: 0.7028 - val_m
Epoch 4/100
194/194 [==============================] - 198s 1s/step - loss: 0.4911 - IOU: 0.6191 - mcc: 0.7248 - dice_coef: 0.7525 - accuracy: 0.9191 - val_loss: 0.4502 - val_IOU: 0.7599 - val_m
Epoch 5/100
194/194 [==============================] - 198s 1s/step - loss: 0.4786 - IOU: 0.6557 - mcc: 0.7546 - dice_coef: 0.7817 - accuracy: 0.9304 - val_loss: 0.4378 - val_IOU: 0.7676 - val_m
Epoch 6/100
194/194 [==============================] - 198s 1s/step - loss: 0.4693 - IOU: 0.6735 - mcc: 0.7689 - dice_coef: 0.7950 - accuracy: 0.9354 - val_loss: 0.4363 - val_IOU: 0.7535 - val_m
Epoch 7/100
194/194 [==============================] - 198s 1s/step - loss: 0.4587 - IOU: 0.6978 - mcc: 0.7904 - dice_coef: 0.8141 - accuracy: 0.9425 - val_loss: 0.4349 - val_IOU: 0.7657 - val_m
Epoch 8/100
194/194 [==============================] - 198s 1s/step - loss: 0.4509 - IOU: 0.7122 - mcc: 0.8010 - dice_coef: 0.8233 - accuracy: 0.9464 - val_loss: 0.4083 - val_IOU: 0.8209 - val_m
Epoch 9/100
194/194 [==============================] - 198s 1s/step - loss: 0.4423 - IOU: 0.7295 - mcc: 0.8155 - dice_coef: 0.8372 - accuracy: 0.9500 - val_loss: 0.4317 - val_IOU: 0.7451 - val_m
Epoch 10/100
194/194 [==============================] - 198s 1s/step - loss: 0.4333 - IOU: 0.7522 - mcc: 0.8332 - dice_coef: 0.8532 - accuracy: 0.9557 - val_loss: 0.3982 - val_IOU: 0.8418 - val_m
```

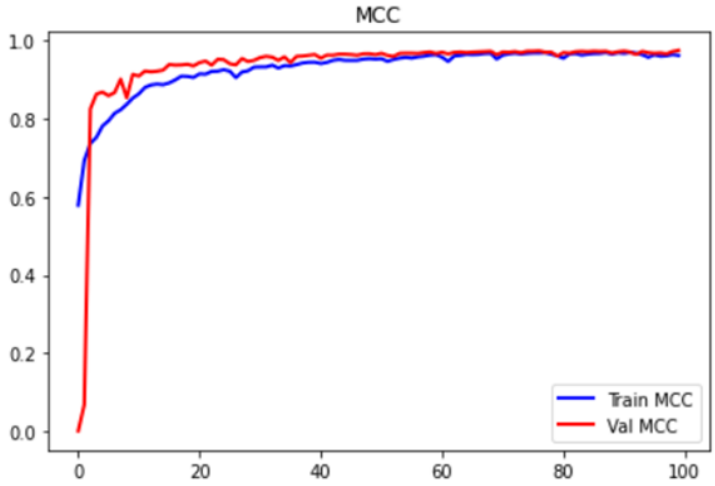
```
Epoch 90/100
194/194 [==============================] - 198s 1s/step - loss: 0.1817 - IOU: 0.9413 - mcc: 0.9645 - dice_coef: 0.9696 - accuracy: 0.9876 - val_loss: 0.2053 - val_IOU: 0.9451 - val_m
Epoch 91/100
194/194 [==============================] - 198s 1s/step - loss: 0.1818 - IOU: 0.9393 - mcc: 0.9628 - dice_coef: 0.9679 - accuracy: 0.9865 - val_loss: 0.2048 - val_IOU: 0.9488 - val_m
Epoch 92/100
194/194 [==============================] - 198s 1s/step - loss: 0.1799 - IOU: 0.9429 - mcc: 0.9657 - dice_coef: 0.9704 - accuracy: 0.9882 - val_loss: 0.2058 - val_IOU: 0.9437 - val_m
Epoch 93/100
194/194 [==============================] - 198s 1s/step - loss: 0.1780 - IOU: 0.9418 - mcc: 0.9638 - dice_coef: 0.9680 - accuracy: 0.9869 - val_loss: 0.2066 - val_IOU: 0.9326 - val_m
Epoch 94/100
194/194 [==============================] - 198s 1s/step - loss: 0.1784 - IOU: 0.9357 - mcc: 0.9598 - dice_coef: 0.9643 - accuracy: 0.9857 - val_loss: 0.2060 - val_IOU: 0.9473 - val_m
Epoch 95/100
194/194 [==============================] - 198s 1s/step - loss: 0.1800 - IOU: 0.9209 - mcc: 0.9516 - dice_coef: 0.9579 - accuracy: 0.9844 - val_loss: 0.2081 - val_IOU: 0.9407 - val_m
Epoch 96/100
194/194 [==============================] - 198s 1s/step - loss: 0.1774 - IOU: 0.9331 - mcc: 0.9590 - dice_coef: 0.9641 - accuracy: 0.9859 - val_loss: 0.2075 - val_IOU: 0.9380 - val_m
Epoch 97/100
194/194 [==============================] - 198s 1s/step - loss: 0.1767 - IOU: 0.9317 - mcc: 0.9561 - dice_coef: 0.9606 - accuracy: 0.9843 - val_loss: 0.2062 - val_IOU: 0.9396 - val_m
Epoch 98/100
194/194 [==============================] - 198s 1s/step - loss: 0.1750 - IOU: 0.9333 - mcc: 0.9570 - dice_coef: 0.9616 - accuracy: 0.9844 - val_loss: 0.2064 - val_IOU: 0.9353 - val_m
Epoch 99/100
194/194 [==============================] - 198s 1s/step - loss: 0.1750 - IOU: 0.9374 - mcc: 0.9601 - dice_coef: 0.9648 - accuracy: 0.9857 - val_loss: 0.2048 - val_IOU: 0.9464 - val_m
Epoch 100/100
194/194 [==============================] - 198s 1s/step - loss: 0.1734 - IOU: 0.9353 - mcc: 0.9587 - dice_coef: 0.9625 - accuracy: 0.9851 - val_loss: 0.2033 - val_IOU: 0.9525 - val_m
```











Performance Metrics	**Training Set**	**Validation Set**
IOU	93.53	95.25
Dice Coefficient	96.25	97.56
MCC	95.87	96.74
Accuracy	98.51	97.83
Loss	0.1734	0.2033

Image number: 0
IOU Score: 0.959441065788269
Dice Coefficent: 0.9793001413345337
MCC: 0.9720539450645447

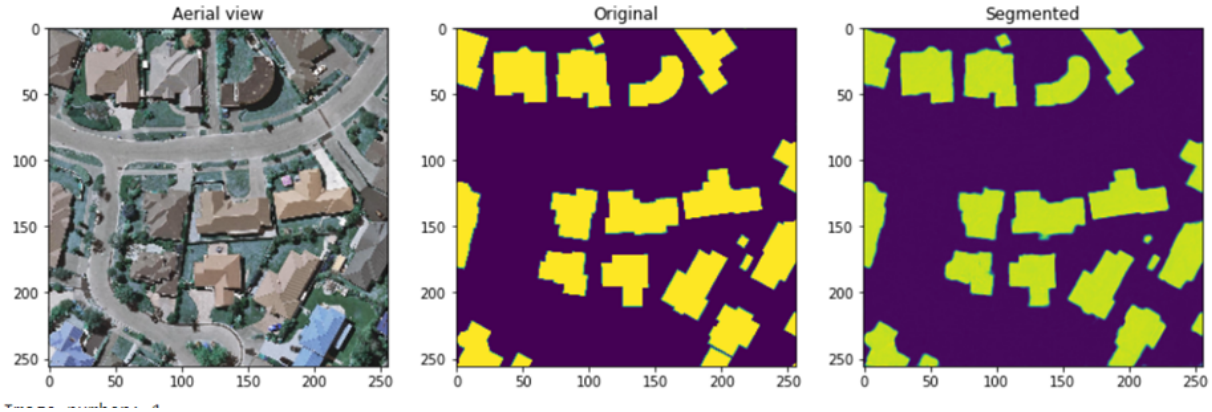


Image number: 1
IOU Score: 0.9527599215507507
Dice Coefficent: 0.975807785987854
MCC: 0.9687467813491821

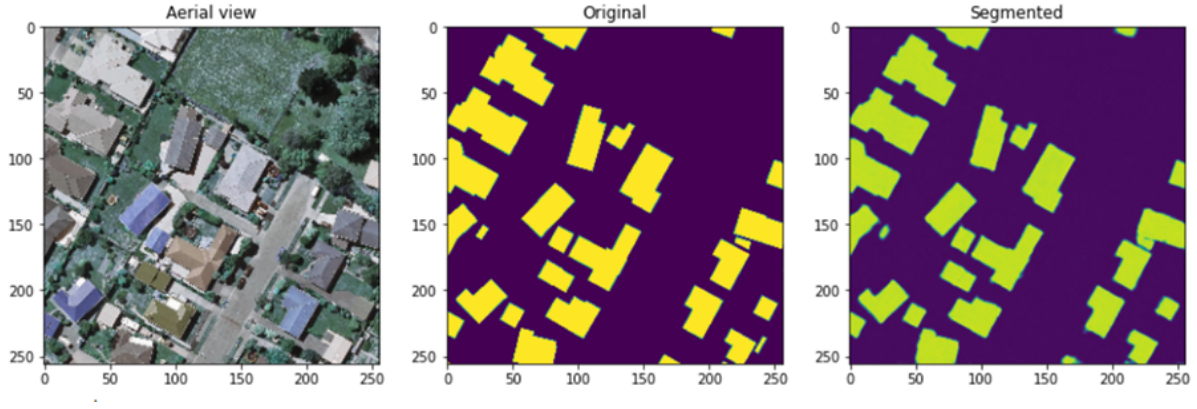
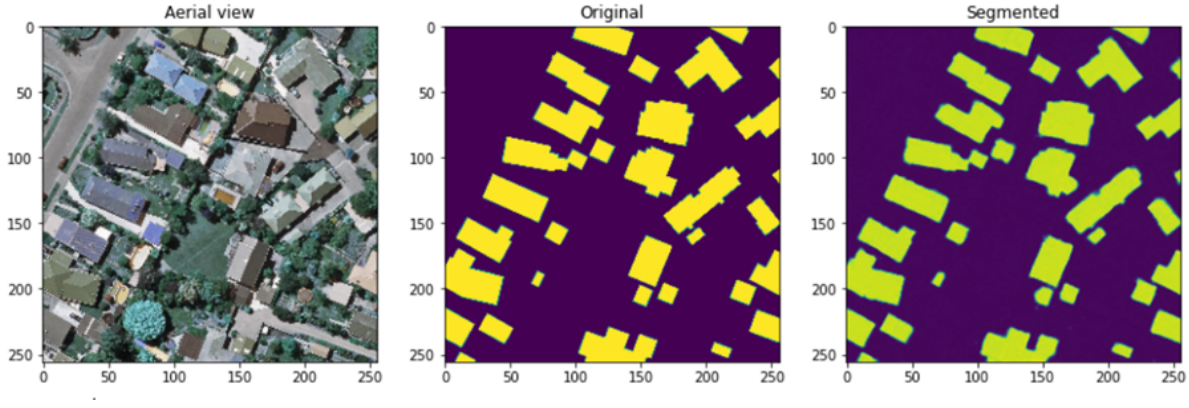
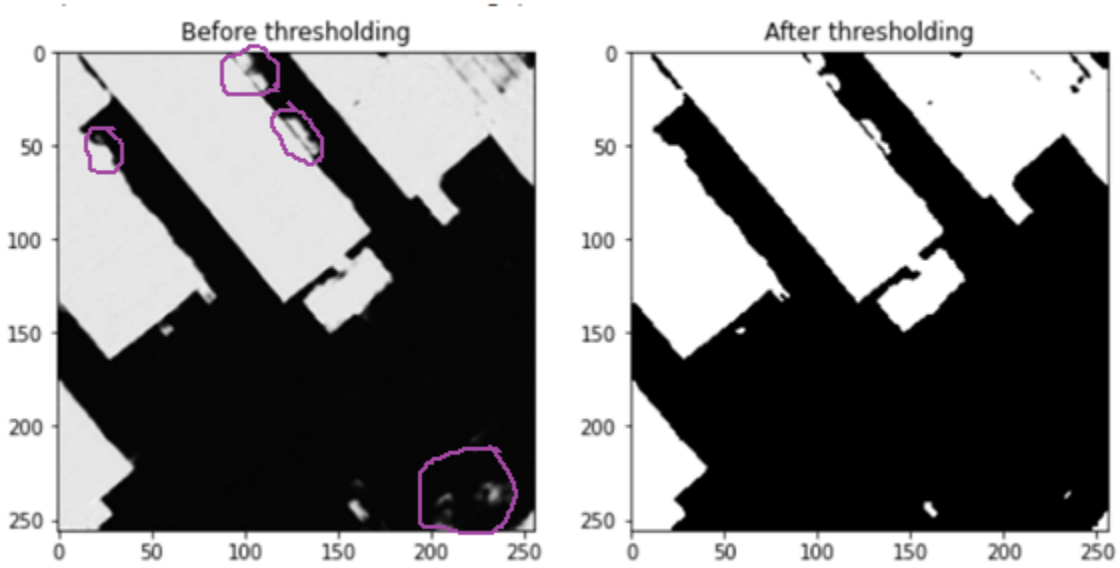


Image number: 2
IOU Score: 0.951558530330658
Dice Coefficent: 0.9751772880554199
MCC: 0.9668542742729187

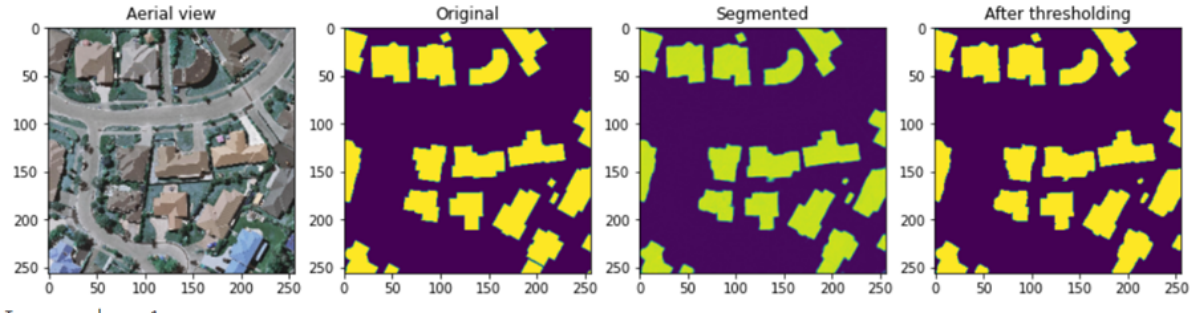


Applying threshold

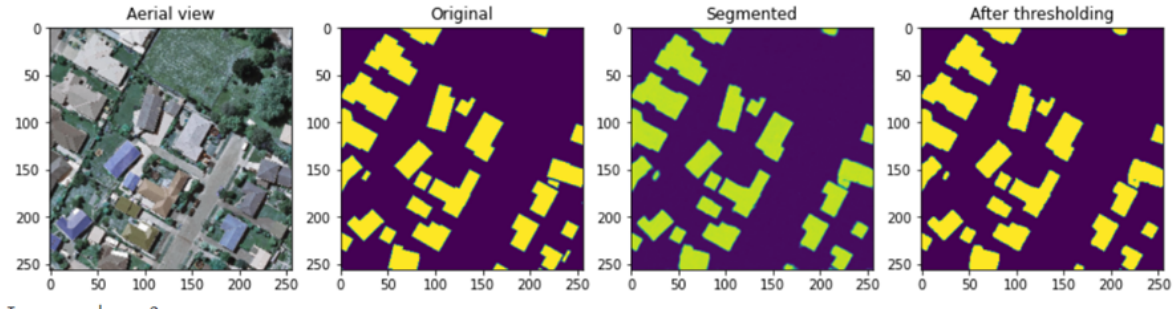
Threshold is set to 0.5 to delineate the boundaries of buildings and differentiate the edges properly. The violet color markings show the difference before and after the threshold is applied.



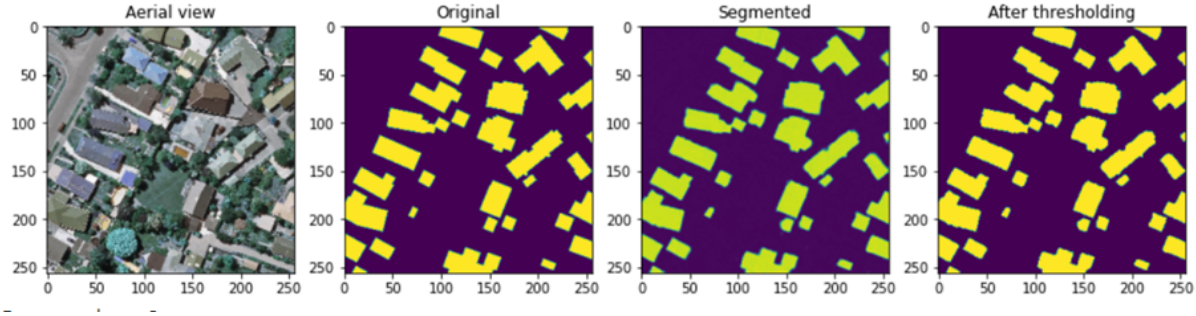

```
Image number: 0
IOU Score: 0.959441065788269
Dice Coefficent: 0.9793001413345337
MCC: 0.9720540046691895
```



```
Image number: 1
IOU Score: 0.9527599215507507
Dice Coefficent: 0.975807785987854
MCC: 0.9687466621398926
```




```
Image number: 2
IOU Score: 0.951558530330658
Dice Coefficent: 0.9751772880554199
MCC: 0.9668542742729187
```



METRICS FOR EVALUATION

For building detection segmentation:

- **IoU - Intersection over Union /Jaccard Coefficient**

To quantify the accuracy of our model to predict size for solar PV arrays, we use Jaccard coefficient which is widely used in prior work to measure the similarity between detected regions and ground truth regions. Jaccard Similarity Index(JSI) measures the similarity for the two sets of pixel data, with a range from 0% to 100%. The higher the percentage, the more precise prediction. It is defined as follows:

$$JSI = \frac{r_d \cap r_g}{r_d \cup r_g}$$

where r_d denotes the masked region for building detection, and r_g indicates the groundtruth region for building segmentation

❑ **DICE Coefficient**

We use DICE coefficient to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. DICE coefficient is 2 times the area of overlap divided by the total number of pixels in both the images.The formula is given by:

$$\frac{2*|X \cap Y|}{|X|+|Y|}$$

where X is the predicted set of pixels and Y is the ground truth.

❑ **MCC - Matthews Correlation Coefficient**

We use the MCC , a standard measure of a binary classifier's performance, where values are in the range −1.0 to 1.0, with 1.0 being perfect building segmentation, 0.0 being random building segmentation, and −1.0 indicating building segmentation is always wrong. The expression for computing MCC is below, where TP is the fraction of true positives, FP is the fraction of false positives, TN is the fraction of true negatives, and FN is the fraction of false negatives, such that TP+FP+TN+FN= 1.

$$\frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

❑ **Accuracy**

Accuracy is the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

For roof type classification:

❑ **Classification Report**

The classification report is used to measure the quality of predictions from a classification algorithm. Precision, Recall and F1 scores are calculated on a per-class basis based on True Positives, True Negatives, False Positives, False Negatives. Here, we calculate the above values for each of the classes, namely: Flat, Gable, Complex.

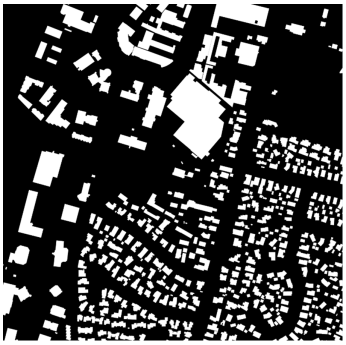
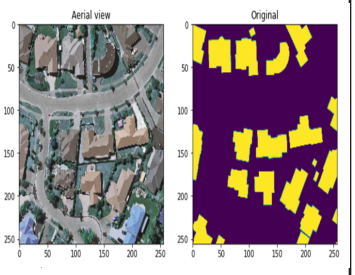
$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \qquad Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

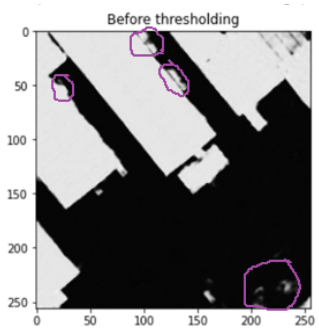
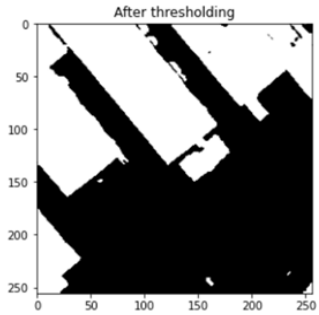
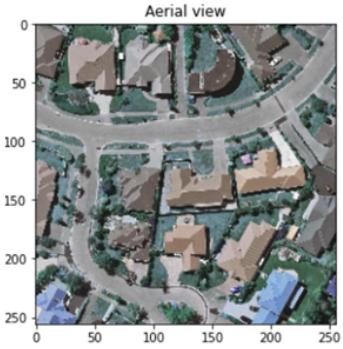
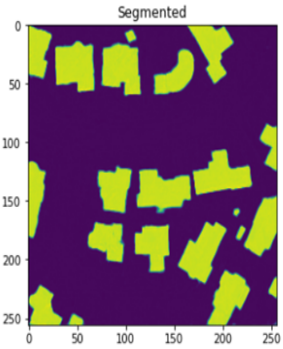
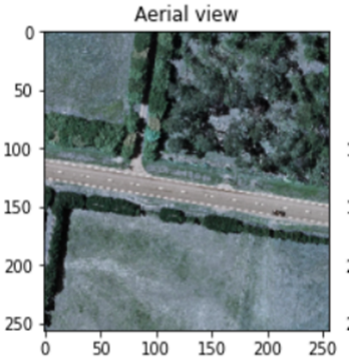
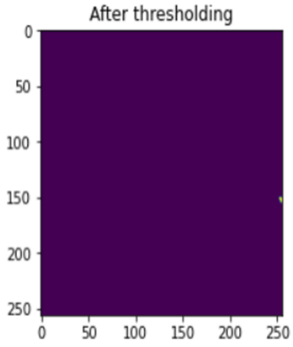
$$F1 = 2.\frac{Precision \times Recall}{Precision + Recall}$$

❑ **AUC-ROC**

The Receiver Operator Characteristic (ROC) is a probability curve that plots the TPR(True Positive Rate) against the FPR(False Positive Rate) at various threshold values and separates the ‘signal’ from the ‘noise’. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes. Higher the values of AUC-ROC, better is the performance of the classification algorithm.

TEST CASES

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_01	Clipping of aerial image is cut in		
TC_02	Clipping of ground truth		
TC_03	Normalization		

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_04	Applying Threshold		
TC_05	Building segmentation of various buildings in a single image		
TC_06	Building segmentation results- with no buildings		

REFERENCES

[1] X. Li, Y. Jiang, H. Peng and S. Yin, "An aerial image segmentation approach based on enhanced multi-scale convolutional neural network," 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS), 2019, pp. 47-52, doi: 10.1109/ICPHYS.2019.8780187.

[2] V. Golovko, S. Bezobrazov, A. Kroshchanka, A. Sachenko, M. Komar and A. Karachka, "Convolutional neural network based solar photovoltaic panel detection in satellite photos," 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2017, pp. 14-19, doi: 10.1109/IDAACS.2017.8094501.

[3] Chen, Mengge and Jonathan Li. “Deep convolutional neural network application on rooftop detection for aerial image.” *ArXiv* abs/1910.13509 (2019): n. Pag.

[4] Kumar, Akash & Sreedevi, Indu. (2018). Solar Potential Analysis of Rooftops Using Satellite Imagery. *ArXiv* abs/1812.11606.

[5] Buyukdemircioglu, Mehmet & Can, Recep & Kocaman, Sultan. (2021). Deep learning based roof type classification using VHR aerial imagery, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLIII-B3-2021. 55-60. 10.5194/isprs-archives-XLIII-B3-2021-55-2021.
[6] B. Chatterjee and C. Poullis, "On Building Classification from Remote Sensor Imagery Using Deep Neural Networks and the Relation Between Classification and Reconstruction Accuracy Using Border Localization as Proxy," 2019 16th Conference on Computer and Robot Vision (CRV), 2019, pp. 41-48, doi: 10.1109/CRV.2019.00014.

[7] Peiran Li, Haoran Zhang, Zhiling Guo, Suxing Lyu, Jinyu Chen, Wenjing Li, Xuan Song, Ryosuke Shibasaki, Jinyue Yan. (2021). Understanding rooftop PV panel semantic segmentation of satellite and aerial images for better using machine learning. Advances in Applied Energy, Elsevier. Volume 4, 100057, ISSN

2666-7924. doi: 10.1016/j.adapen.2021.100057.

[8] Nahid Mohajeri, Dan Assouline, Berenice Guiboud, Andreas Bill, Agust Gudmundsson, Jean-Louis Scartezzini,
A city-scale roof shape classification using machine learning for solar energy applications, Renewable Energy (2018). Volume 121. Pages 81-93. ISSN 0960-1481. doi:10.1016/j.renene.2017.12.096.

[9] Q. Li, Y. Feng, Y. Leng and D. Chen, " SolarFinder: Automatic Detection of Solar Photovoltaic Arrays," 2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2020, pp. 193-204, doi: 10.1109/IPSN48710.2020.00024.

[10] Qi, Chen & Wang, Lei & Wu, Yifan & Wu, Guangming & Guo, Zhiling & Waslander, Steven. (2018). Aerial Imagery for Roof Segmentation: A Large-Scale Dataset towards Automatic Mapping of Buildings. ISPRS Journal of Photogrammetry and Remote Sensing, Elsevier. Volume 147, pp. 42-55.

[11] Edun, Ayobami & Harley, Joel & Deline, Chris & Perry, Kirsten. (2021). Unsupervised azimuth estimation of solar arrays in low-resolution satellite imagery through semantic segmentation and Hough transform. Applied Energy. 298. 10.1016/j.apenergy.2021.117273.