BUILDING EXTRACTION AND ROOF TYPE CLASSIFICATION OF AERIAL IMAGES FOR MAXIMAL PV PANEL INSTALLATION

REVIEW 2 - FINAL YEAR PROJECT (2021 - 2022)

Submitted by

Shruthi M - 2018103592

Gayathri M - 2018103535

Jayapriya M - 2018103029

Under the guidance of

Prof Dr. P. Uma Maheswari

B. E. COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2022

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 detecting the boundaries of rooftop, (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.

A mask R-CNN with three steps is proposed to extract buildings in the city of Christchurch from aerial images post-earthquake in [3] 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.

A combination of image processing techniques, including Adaptive Edge Detection and contours are used by the authors of [4] 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.

The problem of semantic segmentation of buildings from remote sensor imagery is addressed by a novel framework called the ICTNet. 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 et al. (2021) 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.

SSVM based machine learning approach is used to classify building roofs in relation to their solar potential. The SVM classifier in [8] 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

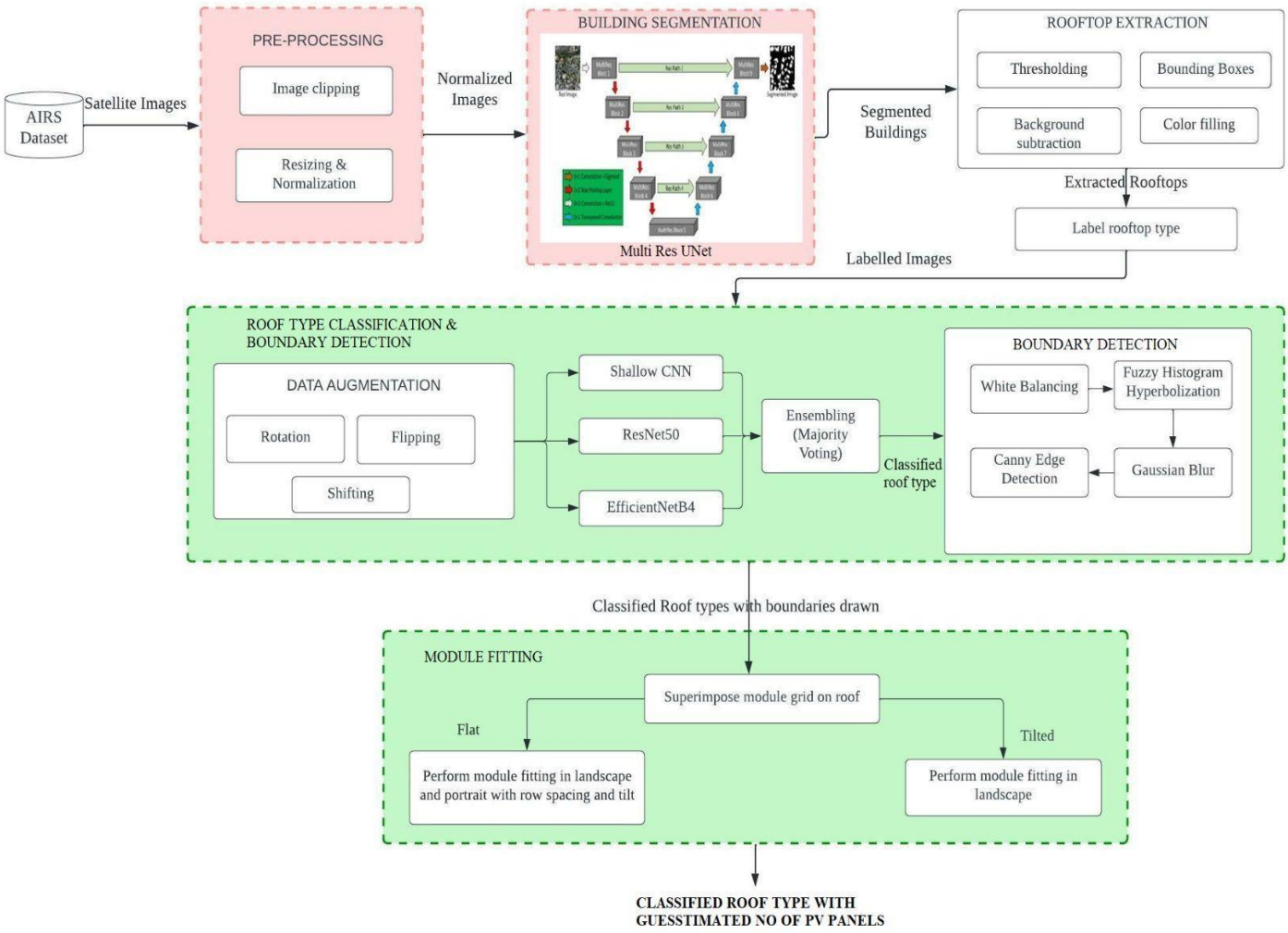


Fig.1 Overall System Architecture

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. An ensemble method (majority voting) is used here to improve the efficiency of classification models. Following that, edge detection of rooftops takes place to mark boundaries on rooftops for fitting PV module grids.

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 in different orientations and find the position that gives maximum count.

MODULE DESIGN

Deep learning based methods, especially UNet models, exhibit great prediction performances on image segmentation tasks. The first module of this project involves building detection and rooftop extraction using MultiRes UNet segmentation. The dataset used here is the AIRS (Aerial Imagery Roof Segmentation) dataset with 857 aerial images in training set and 90 each in testing and validation set with original spatial dimensions of 10000 x 10000 and spatial resolution of 7.5 cm. Given the large dimensions of satellite images, we clip the original images into size of 1536 x 1536. As a result, we use 1548 images for training and 144 images each in the validation and testing set. The extracted rooftop images from the above segmentation are manually labeled into four different classes - namely, Gable, Flat, Complex and Hip. Since this contains around 1000 images, we could further increase the dataset size by augmentation techniques such as rotation, flipping, shifting. Here, the whole process of training and testing the presented network for building detection and rooftop classification is executed under TensorFlow backend and Keras framework in Colab Pro with a memory of 16 GB, T4 GPU. The detailed module design is presented below.

1. BUILDING DETECTION

1.1 *Pre-Processing*

Before building segmentation, a set of pre-processing steps are to be followed. Aerial satellite images from AIRS dataset are of very large dimensions (10000 x 10000). Owing to the hardware constraints for training the model, as a first step in pre-processing we clip the images into smaller images called ‘patches’ or ‘tiles’ to (1536 x 1536) dimensions. By applying this approach, we generate 1548 images for the training set and 144 images each in the testing and validation set. This is followed by resizing the images to 256 x 256 dimensions and further applying MinMax scaler as the normalization technique. Small patches of normalized images are sent to the next layer for model training.

1.2 *Building Segmentation*

Once preprocessing is done, the images are fed into the MuliRes UNet model. The architecture of the model is discussed in the Implementation section below. The

model is trained for 100 epochs and Adam optimizer is used for stochastic gradient descent. After tweaking the hyperparameters, we found that a batch size of 8 with learning rate = 0.0001 works well for our model. The loss function used here is binary cross entropy as we have two classes here - building and the background. ReLu is used as the activation function in the top layers and sigmoid is used as the activation function at the last layer. The segmented masks of buildings are compared with original ground truth masks and performance analysis is carried out.

1.3 *Rooftop Extraction*

The segmented masks of buildings from previous layers undergo thresholding here. Thresholding is done to detect the edges more accurately. After this, contours around the buildings are identified and a bounding box is drawn to the buildings. Following this, background subtraction takes place to extract rooftops from widely dispersed satellite images and they are saved and stored in a new database.

INPUT: AIRS Dataset **OUTPUT:** Rooftops of different buildings

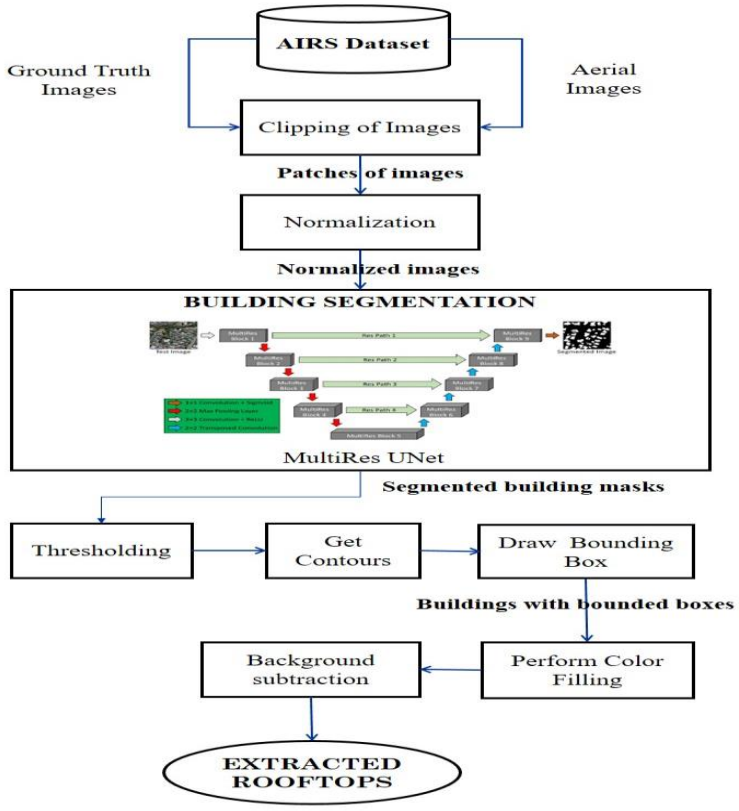


Fig.2 Module Design of Building Detection

2. ROOF TYPE CLASSIFICATION & BOUNDARY DETECTION

2.1 *Data Augmentation*

The extracted rooftops from previous steps are manually labeled into four classes - Flat, Gable, Complex, Hip. Data augmentation is done here to increase the size of the dataset for the roof type classification to increase the accuracy as well as to prevent over-fitting. The techniques used for data augmentation include rotation, shifting, flipping.

2.2 *Classification of roof type images with ensembling approach*

Three deep learning architectures are used here for roof type classification. Shallow CNN is used as a baseline model. Two pre-trained models ResNet50 and EfficientNetB4 are also used and a comparative analysis is made. Further, majority voting is used as an ensembling approach to predict the roof type of unseen images.

2.3 *Boundary Detection*

The extracted roof tops are now further enhanced by applying white balancing to remove haze and then fuzzy histogram hyperbolization takes place. This is followed by applying Gaussian blur and we intend to use Canny Edge Detection for finding the rooftop boundaries.

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

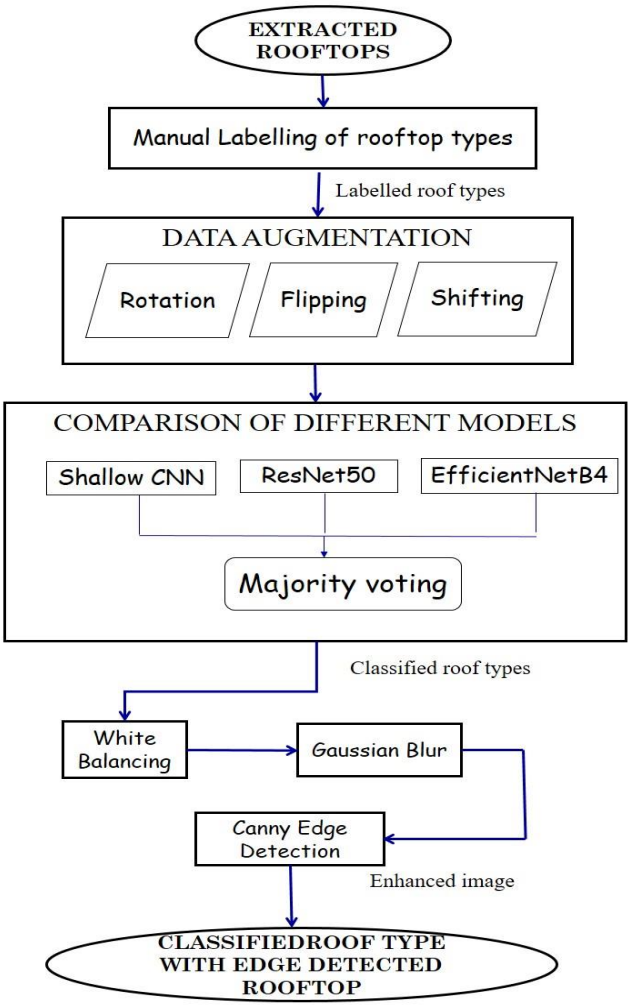


Fig.3 Module Design for Roof type classification & Boundary detection

3. PV MODULE FITTING

The boundary detected rooftops are superimposed with a rectangular PV module shaped grid. Based on the type of roof, module fitting happens. If the type of the roof is flat, we get user input(solar tilt angle and row separation value) and perform maximum fitting algorithm on landscape and portrait mode. If the type of the roof is slope, we perform maximum fitting algorithm on landscape mode. The best fit alignment is chosen and the guesstimated no of PV panels are specified.

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

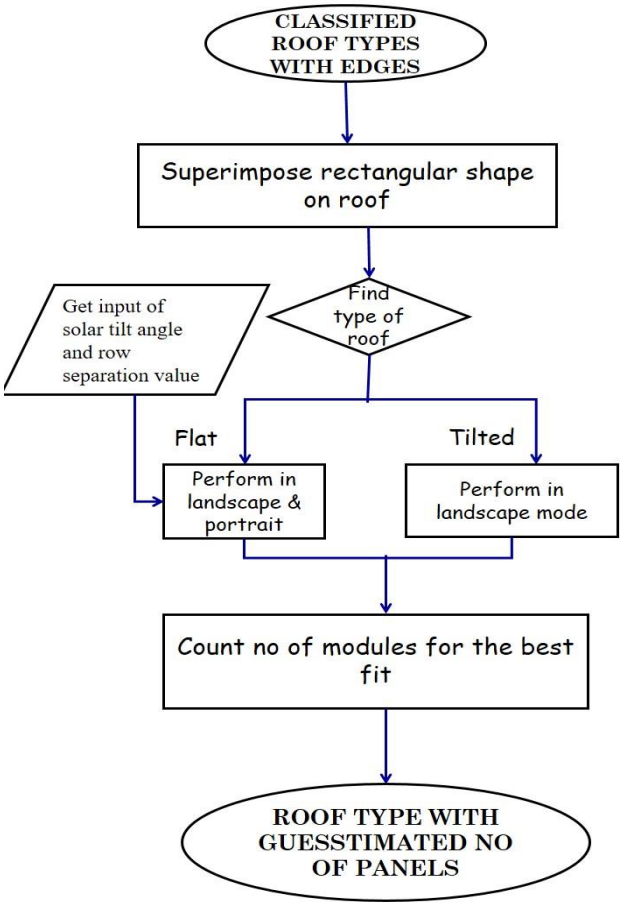


Fig.4 Module Design for PV Module Fitting

IMPLEMENTATION DETAILS

MODULE I: 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.

1.1 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).

Fig.5 Code snapshot for image clipping

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.

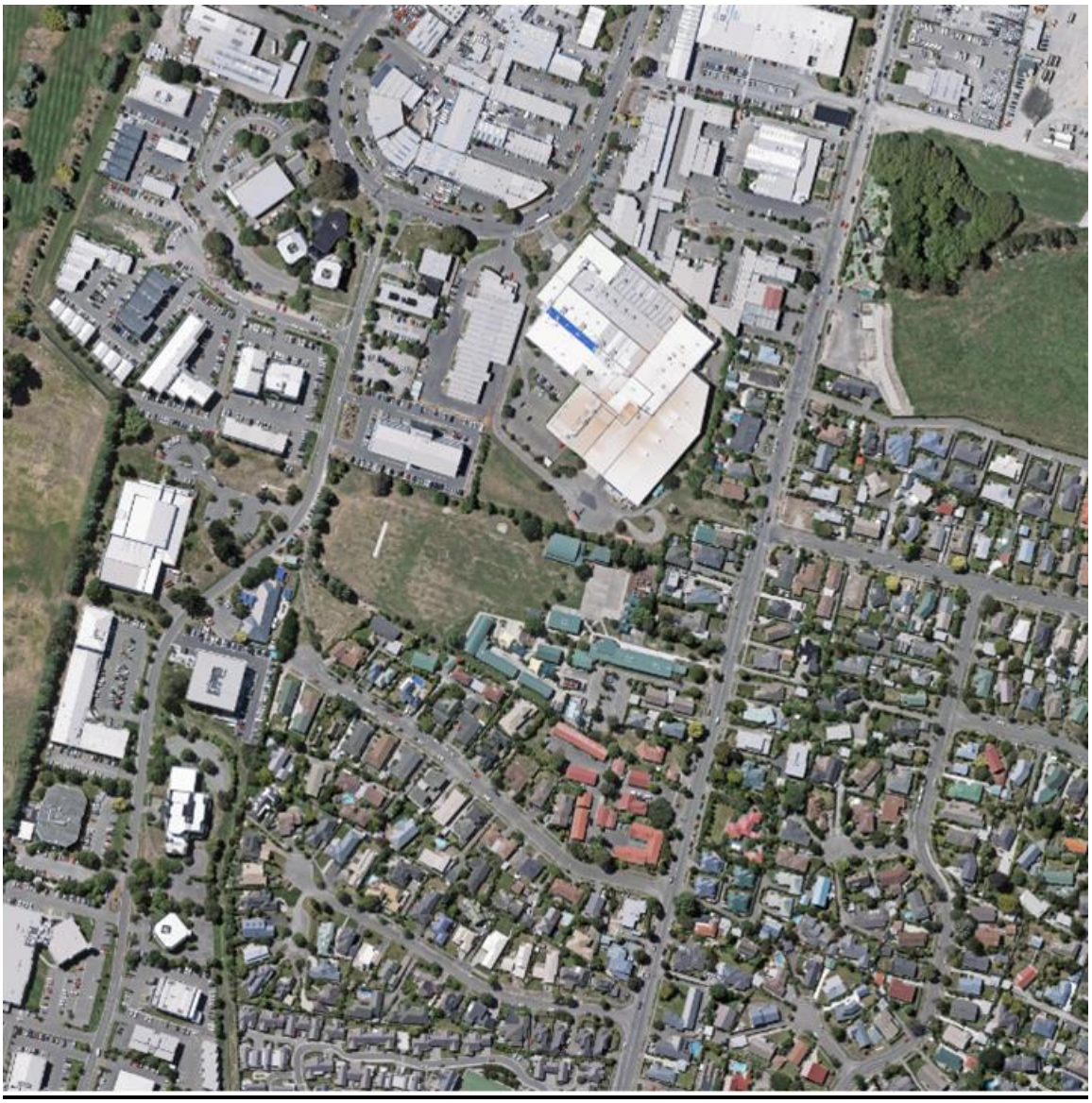


Fig.6 Original Aerial Image

Fig.7 Patches of image after performing clipping on the original image

1.2 Clipping corresponding binary mask images

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

Fig.8 Groundtruth segmented image

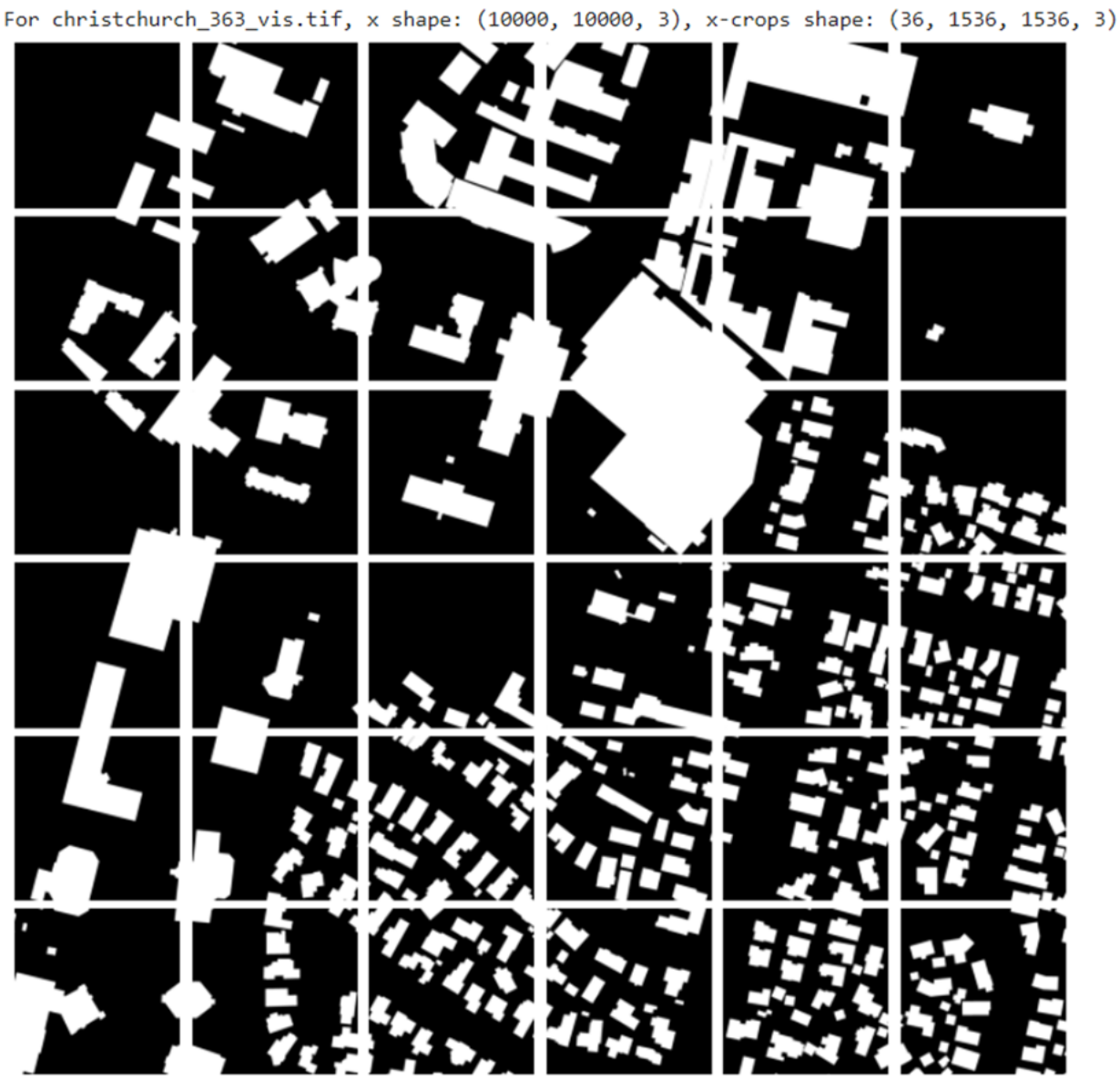


Fig.9 Corresponding patched segmented image after clipping

1.3 Resizing and Normalization

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).

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

Fig.10 Scaling formula

Perform scaling for images using Min Max Scaler

+ Code + Text

```
[ ]  train_X = np.array(train_X)
     train_Y = np.array(train_Y)

     train_Y = train_Y.reshape((train_Y.shape[0],train_Y.shape[1],train_Y.shape[2],1))

     train_X_scaler = scaler.fit_transform(train_X.reshape(-1, train_X.shape[-1])).reshape(train_X.shape)
     train_Y_scaler = scaler.fit_transform(train_Y.reshape(-1, train_Y.shape[-1])).reshape(train_Y.shape)
     print(train_X_scaler.shape, train_Y_scaler.shape)

     (1548, 256, 256, 3) (1548, 256, 256, 1)
```

Fig.11 Code snapshot for image normalization

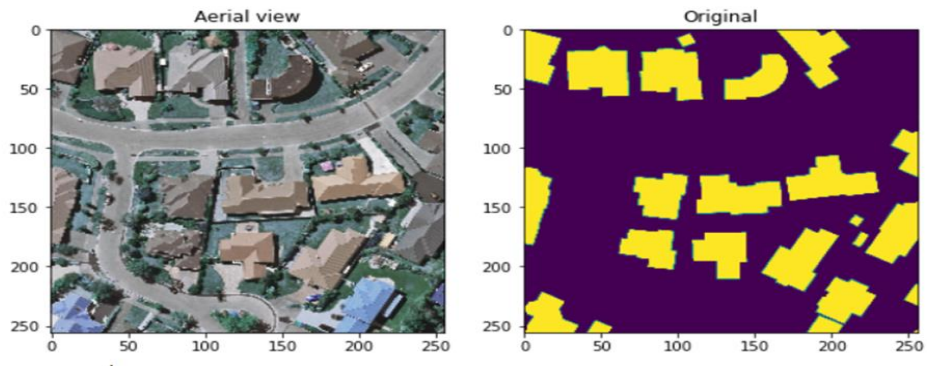


Fig.12 Aerial view and ground truth mask after resizing and employing min-max scaler.

1.4 Adapting the MultiRes UNet architecture

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

1.4.1 MultiRes Block

A sequence of two 3 × 3 convolutional layers is utilized after each pooling layer and transposed convolutional layer in the U-Net design. A single 5 × 5 convolutional operation is equal to a series of two 3 × 3 convolutional processes. As a result, incorporating 3 × 3, 5 × 5, and 7 × 7 convolution operations in parallel is the simplest way to augment U-Net with multi-resolutional analysis, following the strategy of Inception network. By replacing the convolutional layers with Inception-like blocks, the U-Net architecture should be able to reconcile the features learned from the image at different sizes. The addition of more convolutional layers in parallel, on the other hand, dramatically increases the memory required. As a result, we use a sequence of smaller and lighter 3 × 3 convolutional blocks to factorize the larger, more demanding 5 × 5 and 7 × 7 convolutional layers. The outputs of the second and third 3 3 convolutional blocks, respectively, effectively mimic the 5 × 5 and 7 × 7 convolution processes. We hence concatenate the outputs from the three convolutional blocks to extract spatial information from different sizes. Despite the fact that this tweak reduces the RAM requirement significantly, it is still fairly demanding. This is related to the fact that when two convolutional layers are present in a deep network in succession, the number of filters in the first one has a quadratic effect on the memory. Thus rather than keeping all three subsequent convolutional layers with the same number of filters, we gradually increase the number of filters in those layers (from 1 to 3), to prevent the memory requirements of the earlier layers from propagating too far into the network. In order to interpret some more spatial information, we also add a residual link and a 1 1 convolutional layer.

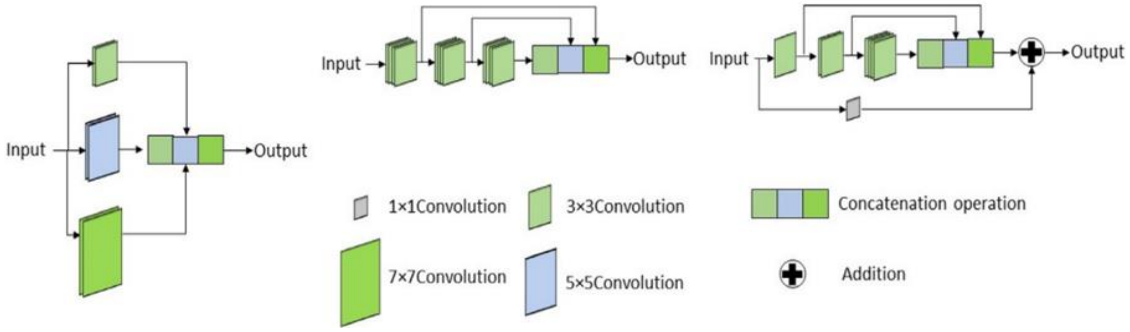


Fig.13 MultiRes Block

1.4.2 Res Path

Skip connections allow the network to transfer spatial information from encoder to decoder that is lost during the pooling procedure. Even if the dispersed spatial elements are preserved, there may be problems. The first shortcut link connects the encoder to the decoder following the last deconvolution step before the first pooling. The encoder's features are expected to be lower level features because they are computed in the network's earlier layers. The decoder characteristics, on the other hand, are expected to be of a considerably higher level, as they are computed at the network's deepest layers. Therefore, they go through more processing. As a result, there may be a semantic gap between the two sets of features that are being merged. This is due to the fact that not only are encoder features going through extra processing, but we're also fusing them with decoder features from much younger layers. Convolutional layers are used along the shortcut links to reduce the discrepancy between the encoder and decoder characteristics. The additional non-linear transformations applied to the features propagating from the encoder stage will account for the additional processing performed by the decoder step. In addition, instead of employing the standard convolutional layers, residual connections are inserted and concatenated with the decoder features.

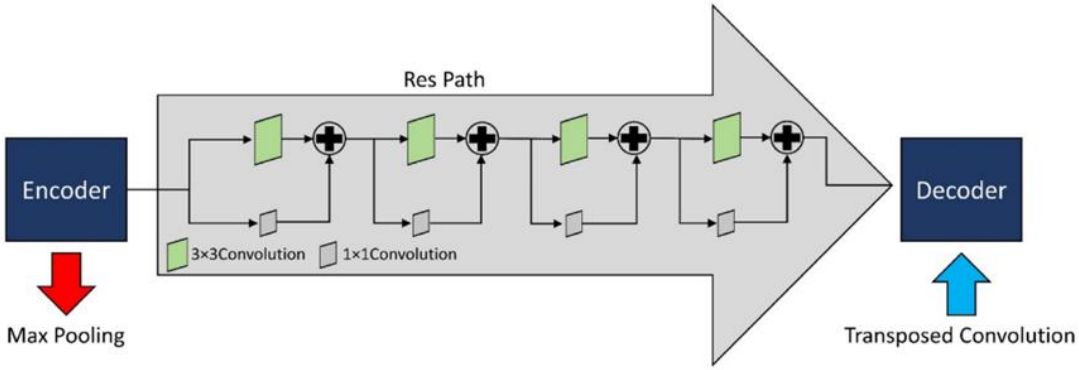


Fig.14 Res Path

1.4.3 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 = $\alpha \times$ □ where α is the scalar coefficient whose value is set to 1.67 and □ refers to the no of filters. The parameter □ 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 □ became double, similar to the original UNet network. The values of □ = [32, 64, 128, 256, 512] are set as follows. We also allocated filters of [/ 6], [□/ 3], and [□/ 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).

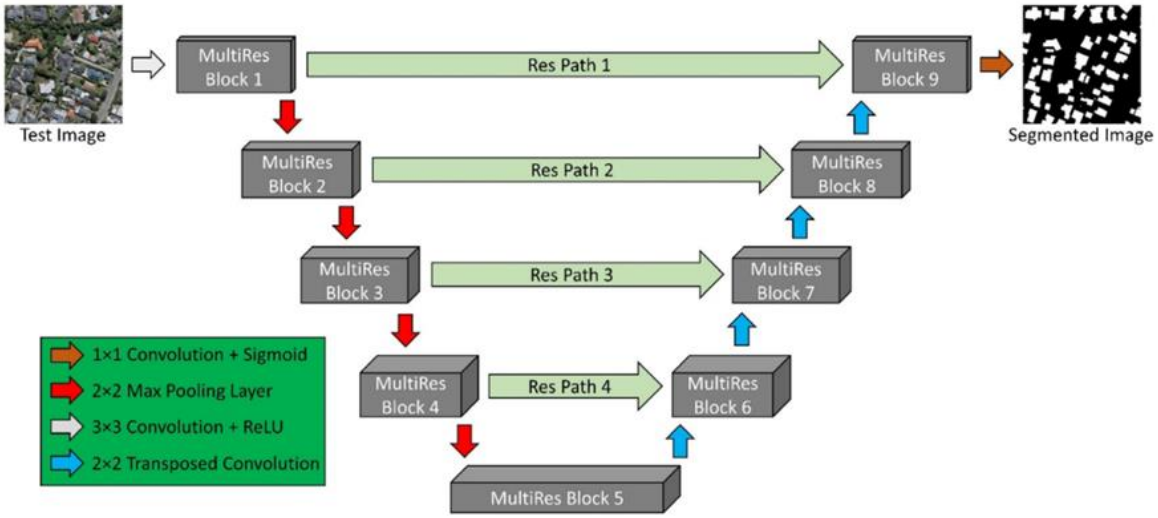


Fig.15 Architecture of MultiRes UNet

```
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
```

Fig.16 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
```

Fig.17 Res Path

Figures 16 & 17 depict the code implementation for MultiRes block and ResPath respectively as described previously. ReLU is used as the activation function here in the top layers except the final output layer as ReLU handles the problem of vanishing gradients and trains the model quicker.

The structure of MultiRes UNet is defined in fig.18. We see there are four levels each in the encoder and decoder part. The number of filters in multires block starts from 32 and increases gradually by multiples of two in the following layers. At each level of the multires block, three convolution operations of strides 3×3, 5×5, and 7×7 are performed followed by max pooling. In the decoder path, transposed convolution takes place at each of the levels. As mentioned in the architecture, the skip connections start with 64 filters at the first level and the number of filters increases by a factor of two. In the final layer, the original dimensions of images (256×256) are restored.

```
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
```

Fig.18 Defining the MultiRes model

```
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
__________________________________________________________________________________________________
```

Fig.19 Brief Model Summary

```
MultiResModel.compile(optimizer = Adam(learning_rate = 0.0001), loss = 'binary_crossentropy', metrics = [IOU, mcc, dice_coef, 'accuracy'], run_eagerly = True)
```

Fig.20 Compiling the MultiRes UNet model

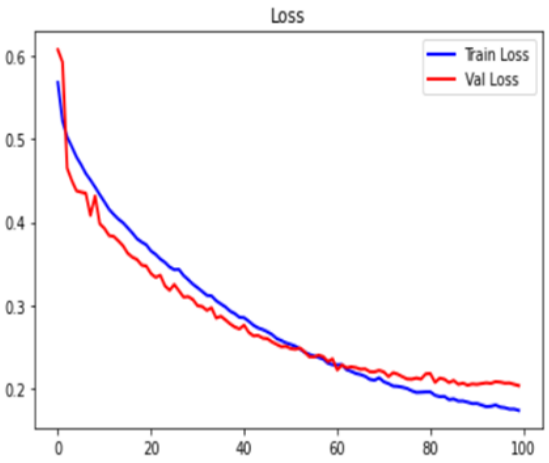
```
[ ] history = MultiResModel.fit(x = train_X_scaler, y = train_Y_scaler, validation_data = (val_X, val_Y), batch_size = 8, epochs = 100, verbose = 1)

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
```

Fig.21 Training the model for 100 epochs

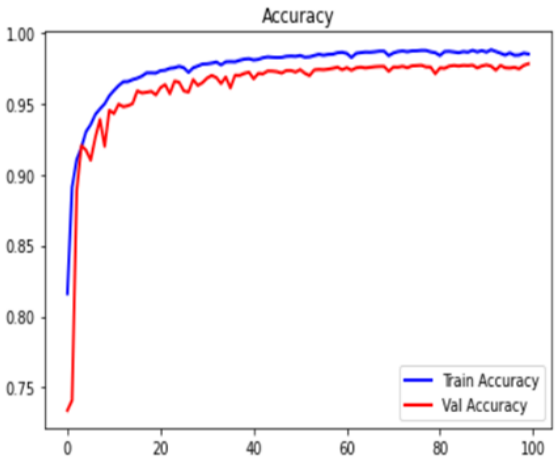
The images have been resized to 256 × 256 dimensions as mentioned in the previous step. After tweaking out the hyperparameters, we found out the following values to give the best results for our model. The model training started with an initial IoU value of 42% and gradually increased to 75% in the next 10 epochs. From the above training, we see that there is no generalization gap between validation IoU and training IoU, and hence the model is not overfitting. In a similar way, the loss value has reduced significantly from 0.5220 in the first epoch to 0.1734 in the final epoch. We also notice MCC and Dice Coefficient to follow similar trends.

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)

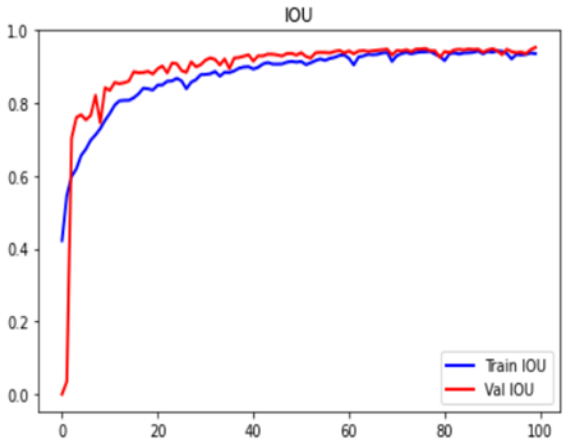
Table 1 Hyperparameter values used in training our MultiRes UNet model



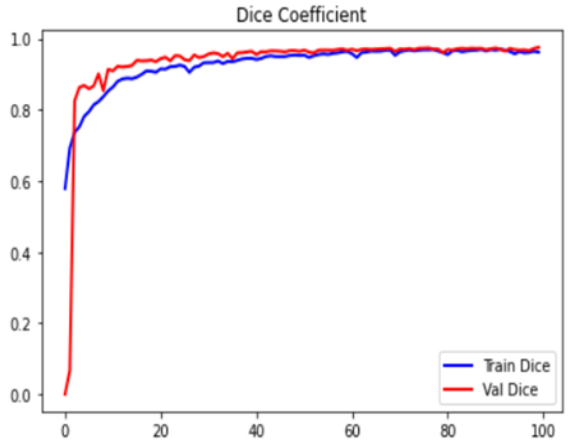
(a) Model Loss



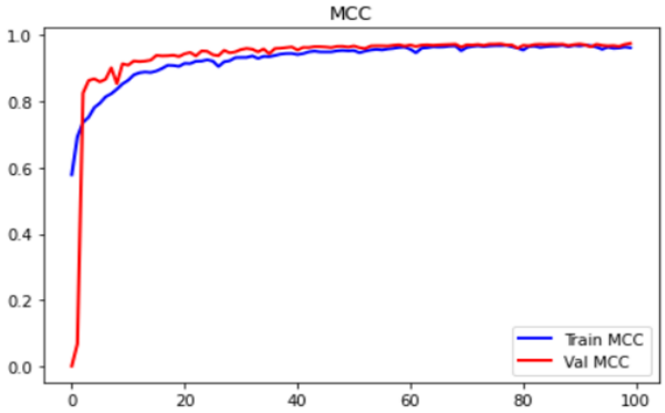
(b) Model Accuracy



(c) Model IOU



(d) Model Dice Coefficient



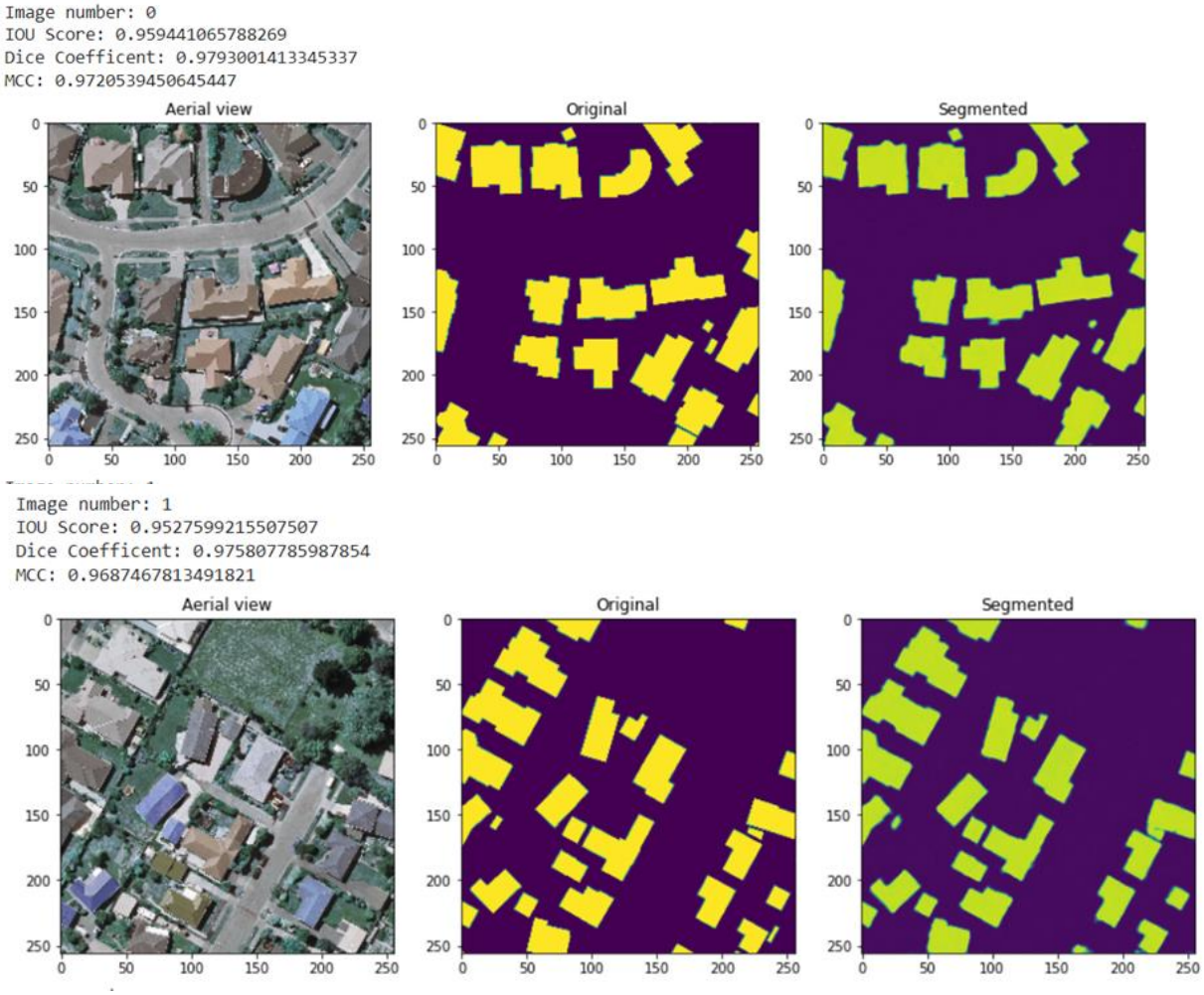
(e) Model MCC

Fig.22 Graphs of the various performance metrics

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

Table 2 Performance results of the model on training and validation set

The graphs in fig.22 allows us to easily interpret the training of the MultiRes UNet model. The blue line indicates the training details while the red line shows the testing details. The MCC value is close to 100% at the end of 100 epochs and so is the dice coefficient value. The difference between training and validation loss is very meager which shows the model does not overfit. Fig.23 compares ground truth images with our segmented images. The results show that on an average an IoU score of 95% is obtained and we notice that even small buildings are correctly segmented by the trained model.



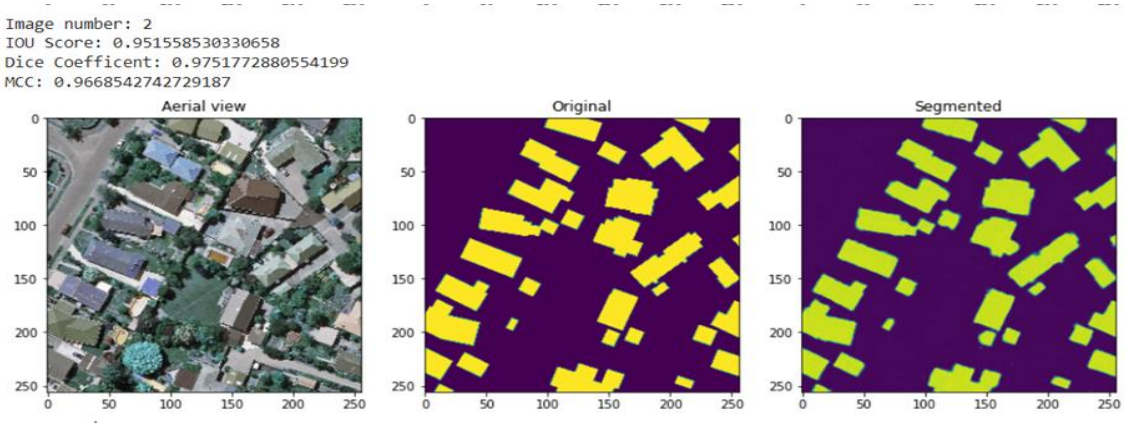


Fig.23 Results of building segmentation with MultiRes UNet model

1.5 Applying threshold

Thresholding is a method where pixels are partitioned based on their intensity value. Here, pixel value 0 indicates black color while pixel value 1 indicates white color. After training the images with the MultiRes UNet model, some pixel values lie in the range between (0,1) and these pixels do not clearly indicate whether they belong to foreground or background. In order to resolve this issue and delineate the boundaries of buildings and differentiate the edges properly, we use the technique of simple thresholding. The threshold value is set to 0.5, and all values greater than 0.5 are assigned to the foreground, while values less than 0.5 are assigned to the background. In fig.24, the violet color indications show the difference in segmentation before and after the threshold is applied. We can see how the images are clearly defined after applying thresholds in the image below. This aids in the separation of two structures that are quite close to one another.

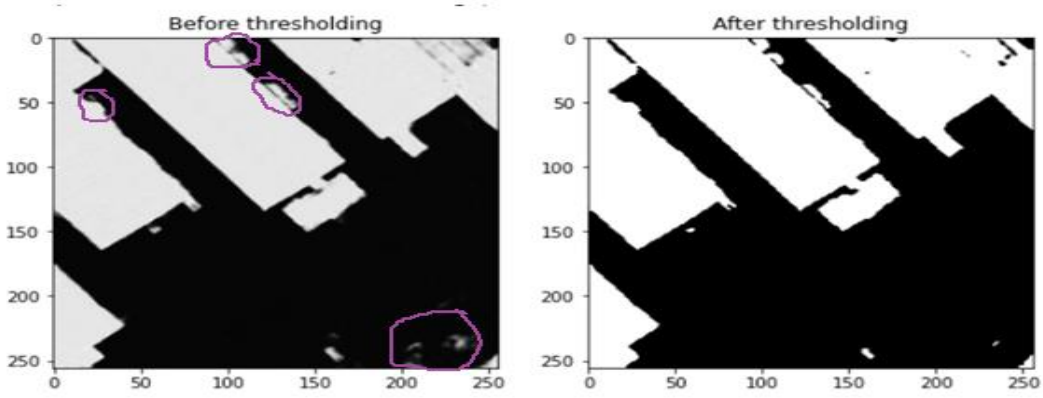


Fig.24 Comparison of results before and after applying threshold

The following figure provides a comparison between the ground truth images with our prediction masks before and after applying threshold. After thresholding, our buildings are more effectively demarcated.

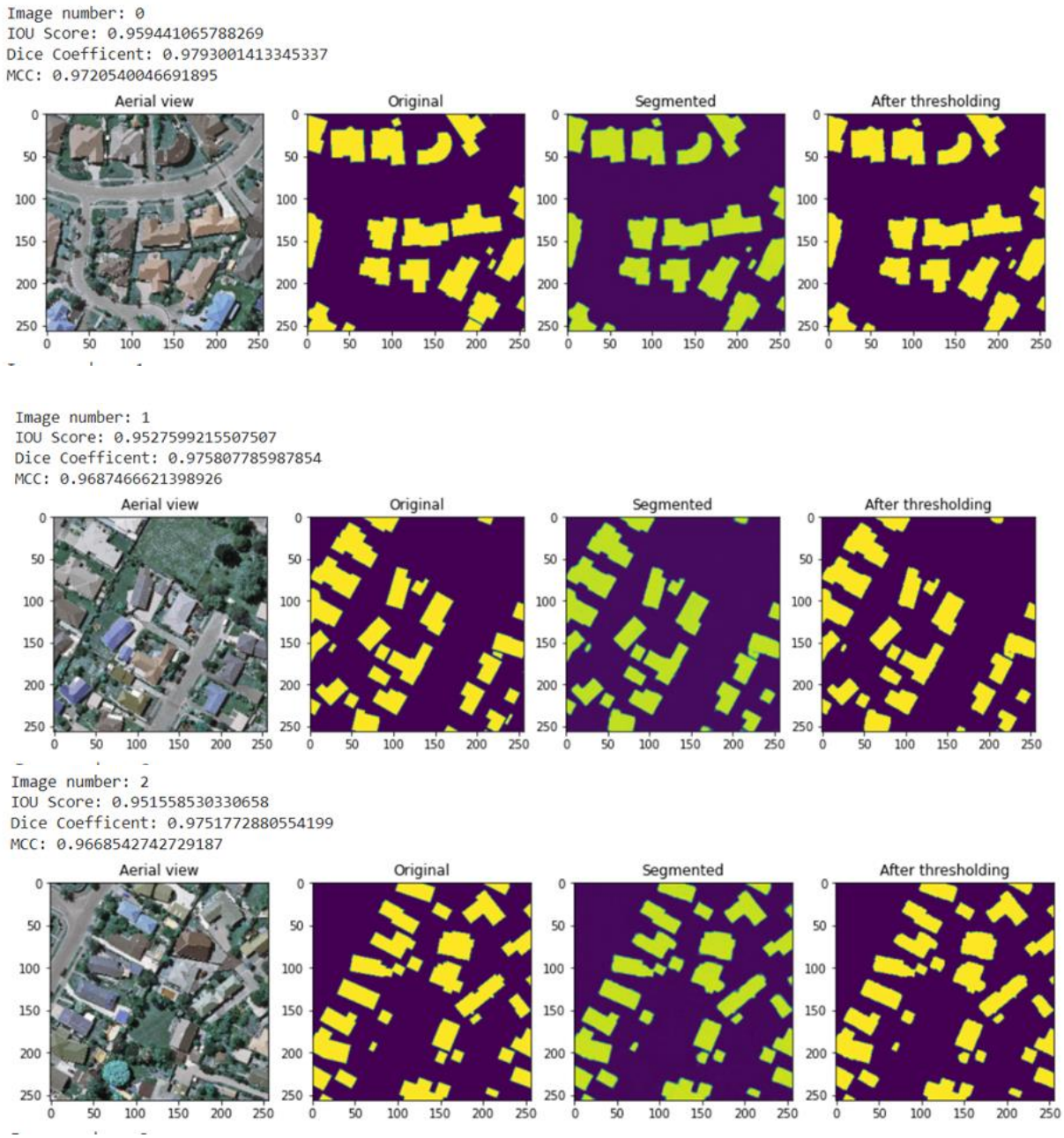


Fig.25 Comparison of ground truth images with our predictions before and after applying threshold

1.6 Rooftop Extraction

1.6.1 Get contours

Contours are curves that connect all continuous points of the same hue or intensity. They are used to detect the borders of objects and localize them easily in an image. The segmented building masks are first transformed to grayscale and then binary thresholding is applied on it following which the contours of buildings are identified. For each image, the number of buildings in the given aerial satellite image is identified by this method.

We use the findContours function to draw contours around the detected buildings. The mode used here is cv2.RETR_EXTERNAL which retrieves external or outer contours of the building. The algorithm used for contour detection is CHAIN_APPROX_SIMPLE that removes all redundant points and compresses the contour, thereby saving memory. The result is the detected contours where each contour is stored as a vector of points.

Draw bounding boxes on the binary segmented mask & perform color filling

- Convert the image to grayscale.
- Apply threshold.
- Get contours.
- Fill bounding box with color
- Store {x,y,w,h} in a vector of list

```
# read image
img_mask = cv2.imread(train_path_label)
print(type(img))
# convert to grayscale
gray = cv2.cvtColor(img_mask,cv2.COLOR_BGR2GRAY)

# threshold
thresh = cv2.threshold(gray,128,255,cv2.THRESH_BINARY)[1]

# get contours
result = img_mask.copy()
seg_with_bounded_box = img_mask.copy()
contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
print("No of identified buildings: {0}".format(len(contours)))
bb = []
```

Fig.26 Code implementation of drawing contours

1.6.2 Draw bounding boxes

Bounding boxes are used to highlight the regions of interest in the image. The detected contour is enclosed within a rectangle using the boundingRect function. The top-left (x,y) coordinates of the bounding box are found along with the width and height of each box. Each of the bounding box coordinates are stored in a vector. For the purpose of drawing rectangle around the segmented masks, the cv2.boundingRect() function is used. This function is used mainly to highlight the region of interest after obtaining contours from an image.

```
# draw bounding boxes
for cntr in contours:
    x,y,w,h = cv2.boundingRect(cntr)
    cv2.rectangle(result, (x, y), (x+w, y+h), (0, 0, 255), 2)
    cv2.rectangle(seg_with_bounded_box, (x, y), (x+w, y+h), (255, 255, 255), -1)
    # print("x,y,w,h:",x,y,w,h)
    bb.append([x,y,w,h])

print(bb)
```

```
<class 'numpy.ndarray'>
No of identified buildings: 36
[[4398, 9761, 57, 48], [2593, 6474, 179, 134], [2794, 6393, 39, 47], [2518, 6162, 300, 227], [4495, 5982, 48, 49], [4548, 5860, 140, 123], [3544, 5015, 81, 73],
```

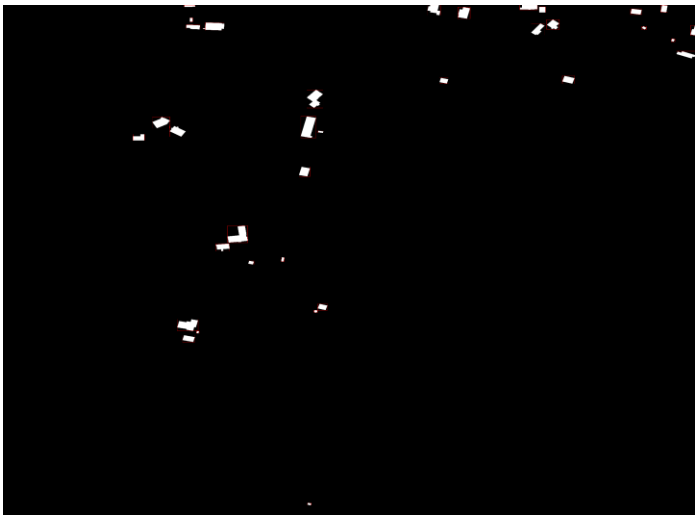


Fig.27 Bounding boxes drawn around the segmented mask images

After performing building detection, we draw bounding boxes as mentioned above and the above figure is an illustration where rectangular boxes are drawn after finding the contours.

1.6.3 Background Subtraction

Background subtraction refers to a technique where an image's foreground is extracted for further processing like object recognition, classification. In our case, the foreground is the detected buildings with contours as found in the previous step (fig.27). The aerial satellite image is subtracted from the bounding box drawn image and rooftops are extracted.

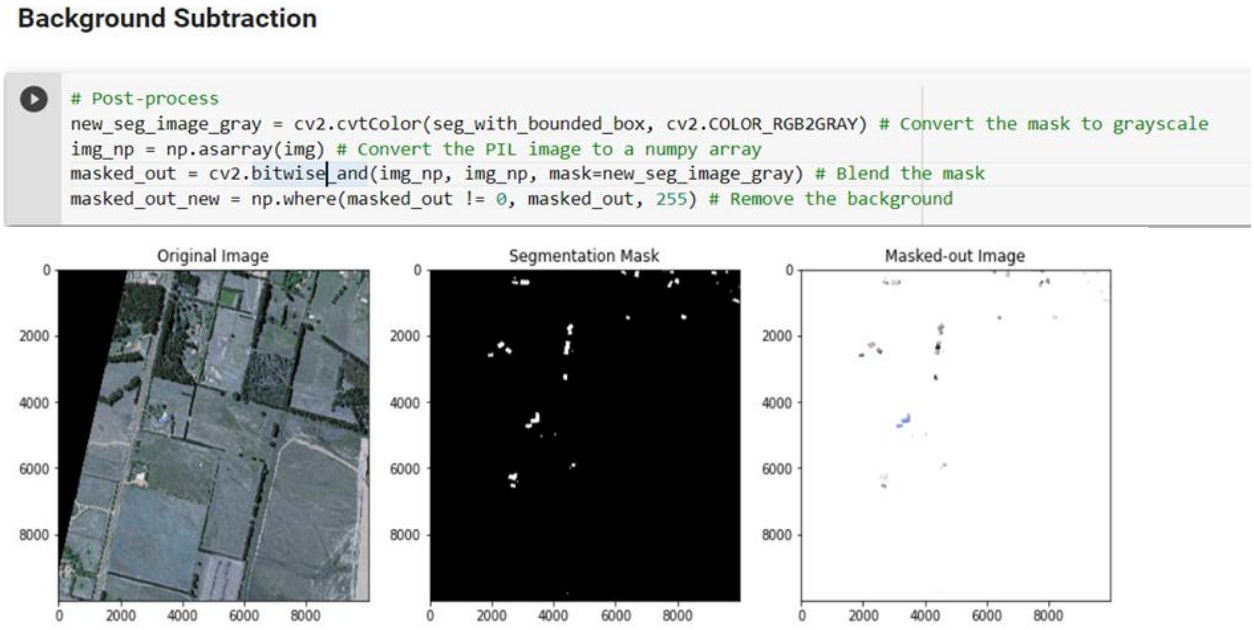


Fig.28 Original satellite image (left), masked image after training on MultiRes UNet model (center), extracted rooftops (right)

The rooftops are extracted by the above process and are stored in a database which is later used in the second stage for classification.

MODULE II: ROOF TYPE CLASSIFICATION AND BOUNDARY DETECTION

2.1 Preparation of dataset and labeling roof type

The second phase in the pipeline is classifying the type of roofs and drawing boundaries on the rooftop. After conducting rooftop extraction in the preceding step, a dataset including 1115 rooftop photos is populated into three different categories: Flat, Gable, and Hip. We had to manually label the data as there was no supervised dataset for roof type classification available. After reading numerous publications, we discovered that the authors used manual roof type labeling in every case. The task of manual labeling does not require any prior knowledge of the field and roof types in most cases can be easily identified by laymen. Thus, we referred to the sample images provided in [5] for labeling our dataset.

The dataset was cleaned up by removing roof types with poor resolution. A balance between the several roof type classifications was sought as much as feasible when compiling the dataset. Despite this, the number of rooftops of type gable is considerably higher than the other two classes due to the varying numbers of instances for each roof type in the research area.
The distribution of different roof types are as follows:

Roof Type	**Training** (91%)	**Validation** (5%)	**Testing** (4%)	**Total**
Flat	303	14	18	335
Gable	410	23	18	451
Hip	307	13	9	329

Table 3 Distribution of roof type dataset

FLA			
GA			
HIP			

Table 4 Samples of images of each class from Christchurch, New Zealand

2.2 Data Augmentation

For training neural network models, the size of the dataset needs to be large. However, owing to time constraints we were able to manually label only 1115 images. As a result, we employ data augmentation approaches to increase the dataset size for better model training. Rotation, shifting and flipping are the three main techniques used here.

(i) Rotation - This is used to randomly rotate images in the range present. We set the rotation range to 7 degrees here.

(ii) Flipping - In flipping, images are randomly flipped in both horizontal and vertical direction.
(iii) Shifting - Images are shifted randomly based on the percentage mentioned in horizontal and vertical direction.

Data Augmentation

```
data_generation = ImageDataGenerator(
    rotation_range = 7,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range = 0.10,  # randomly shift images horizontally (fraction of total width)
    height_shift_range = 0.10,  # randomly shift images vertically (fraction of total height)
    horizontal_flip = True,  # randomly flip images
    vertical_flip = True,   # randomly flip images
    fill_mode = 'reflect')
data_generation.fit(train_X)
```

Fig.29 Implementation of data augmentation techniques

The below figure shows horizontal flipping and width shifting applied to rooftop images.

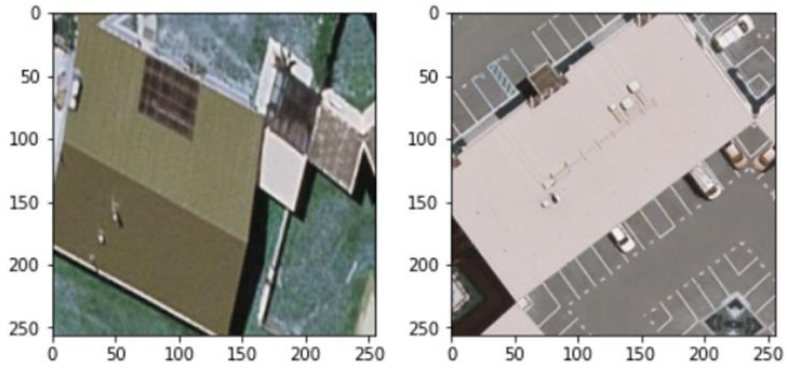


Fig.30 Results of image after performing data augmentation

2.3 Roof type classification

Following data augmentation, the model must be trained to recognise different types of roofs. We use the CNN model as well as additional transfer learning models like VGG16, ResNet50, and EfficientNetB4 to train. After the models have been trained, we employ majority voting as our ensembling approach to combine learnings from different models and for a better result.

2.3.1 Shallow CNN Model

The shallow CNN model used here has 6 convolution layers and 6,751,809 trainable parameters. For the convolutional layers, a 3x3 kernel filter size was selected (Conv2D). In the pooling layers, the default pool size (2x2) was employed (MaxPooling2D). A dropout of 0.2 is used to prevent the model from overfitting. ReLU is used for the activation function with categorical cross entropy as the loss function. The detailed model summary is present in the following figure. Both RMSProp and Adam optimizers were tried for training the model and RMSProp gave better results. With a learning rate of 0.0001 and a batch size of 16, the model was trained for 100 epochs.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 64)      1792
_________________________________________________________________
activation (Activation)      (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 254, 254, 110)     63470
_________________________________________________________________
activation_1 (Activation)    (None, 254, 254, 110)     0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 110)     0
_________________________________________________________________
dropout (Dropout)            (None, 127, 127, 110)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 127, 127, 84)      83244
_________________________________________________________________
activation_2 (Activation)    (None, 127, 127, 84)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 125, 125, 84)      63588
_________________________________________________________________
activation_3 (Activation)    (None, 125, 125, 84)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 84)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 62, 84)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 62, 62, 64)        48448
_________________________________________________________________
activation_4 (Activation)    (None, 62, 62, 64)        0
_________________________________________________________________
```

```
conv2d_5 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
activation_5 (Activation)    (None, 60, 60, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 30, 30, 32)        18464
_________________________________________________________________
activation_6 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
activation_7 (Activation)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              6423552
_________________________________________________________________
activation_8 (Activation)    (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 3075
_________________________________________________________________
activation_9 (Activation)    (None, 3)                 0
=================================================================
Total params: 6,751,809
Trainable params: 6,751,809
Non-trainable params: 0
```

Fig.31 Model summary of CNN model

2.3.4 VGG

VGG16 is one of the most famous transfer learning models trained on the ImageNet database. The architecture of VGG goes as follows: A stack of multiple convolution layers of filter size 3 x 3, stride one, and padding 1, followed by a max-pooling layer of size 2 x 2, is the basic building block for all of these configurations. There are a total of 13 convolutional layers and 3 fully connected layers in VGG16 architecture. Image is passed through the first stack of 2 convolution layers of the very small receptive size of 3 x 3, followed by ReLU activations. Each of these two layers contains 64 filters. The activations then flow through a similar second stack, but with 128 filters as against 64 in the first one. This is followed by the third stack with three convolutional layers of kernel size 256 and a max pool layer. This is followed by two stacks of three convolutional layers, with each containing 512 filters.

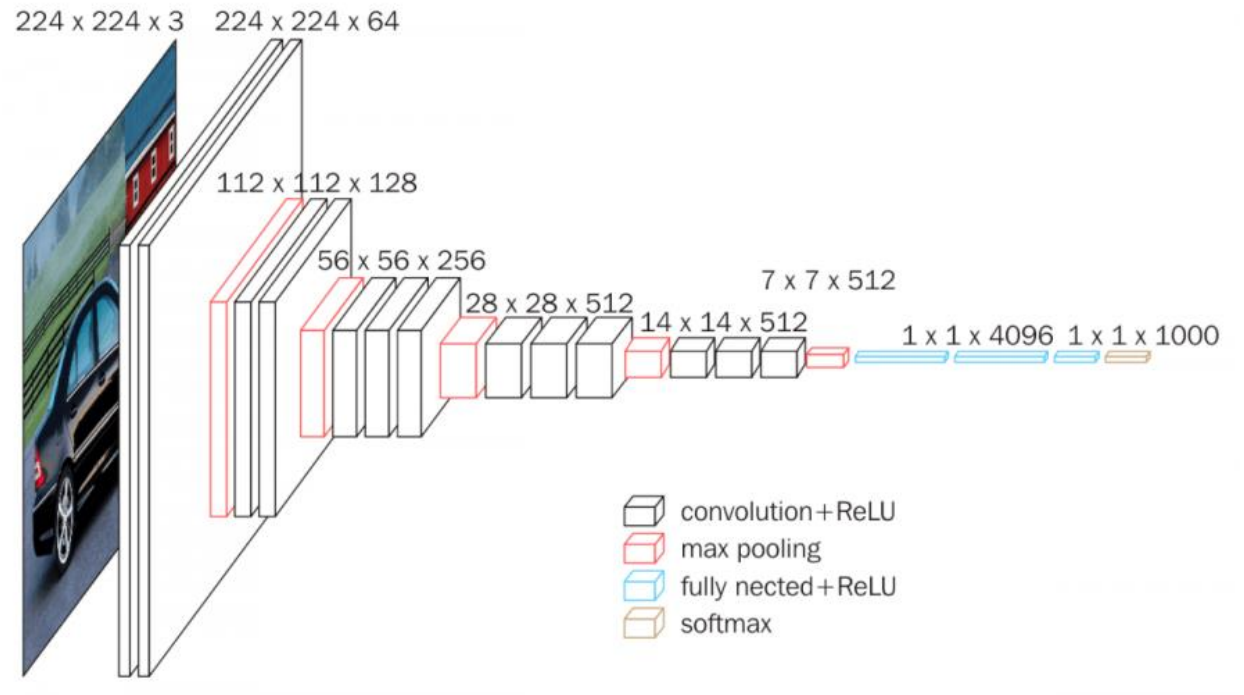


Fig.32 Architecture of VGG16 model

In our project, the fully connected layers of the 3 networks are replaced with the fully connected layer block with 64, 128, 256, 512 nodes. The last layer has three nodes representing the three classes for our classification. The model is trained for 50 epochs with a learning rate of 0.0001 and batch size of 4. RMSProp Optimizer is used here.

RESNET50

The architecture of ResNet50 contains a convolution with a kernel size of 7 * 7 and 64 different kernels all with a stride of size 2 giving us 1 layer. This is followed by max pooling with a stride size of 2. In the next convolution there is a 1 * 1, 64 kernel following this a 3 * 3, 64 kernel and at last a 1 * 1, 256 kernel. These three layers are repeated a total of 3 times so giving us 9 layers in this step. The entire flayer model is present in the diagram below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig.33 Architecture of ResNet50 model

The model is trained for 50 epochs with a learning rate of 0.0001 and batch size of 8 and RMSProp optimizer is used.

EFFICIENTNETB4

EfficientNetB4 is yet another best pre-trained model. There are five modules in total in EfficientNetB4. Module 3 is used as a skip connection to all the sub-blocks and module 4 for combining the skip connections in the first sub-blocks.

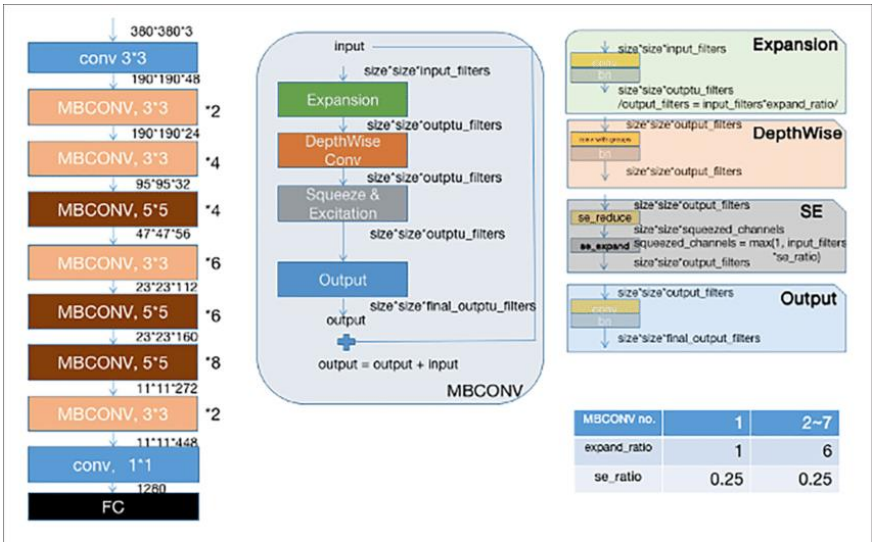


Fig.34 Architecture of EfficientNetB4 model

All the layers after top_bn are set to trainable. Totally, 125,207 are non-trainable parameters. The model is trained for 50 epochs with a learning rate of 0.0001 and batch size of 8. EfficientNet works well with Adam Optimizer and hence is used here.

2.4 Performance results

2.4.1 Performance of Shallow CNN Model

The CNN model achieved an accuracy of 78% on the training set and 74% on the validation set. From the below graphs, it can be inferred that there is no generalization gap because of dropout and data augmentation. The model was likewise trained for 50 epochs at first, but because it didn't produce satisfactory results, the model was further trained for another 50 epochs.

The confusion matrix reveals that gable and hip classes are accurately classified to a larger extent than other classes. An equal number of flat classes, on the other hand, are misclassified as gables. The results also show that flat class has the highest precision value of 75% followed by hip and then gable. The overall accuracy on the test samples accounts to 69%.

contradicting

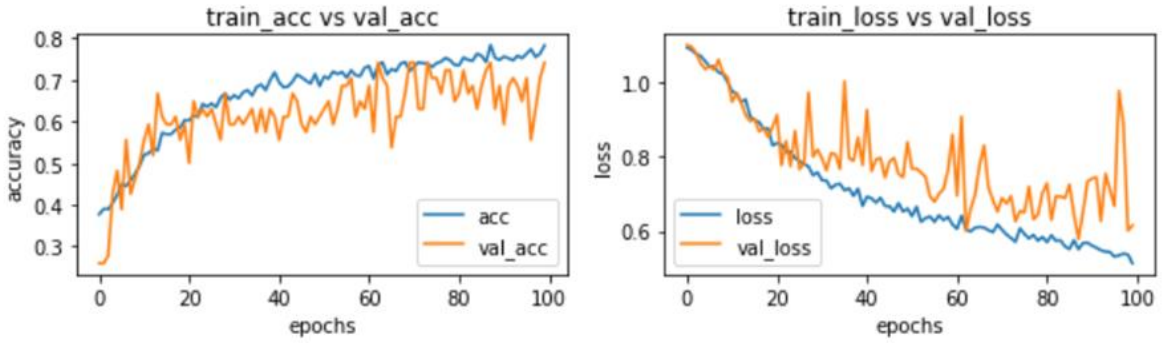
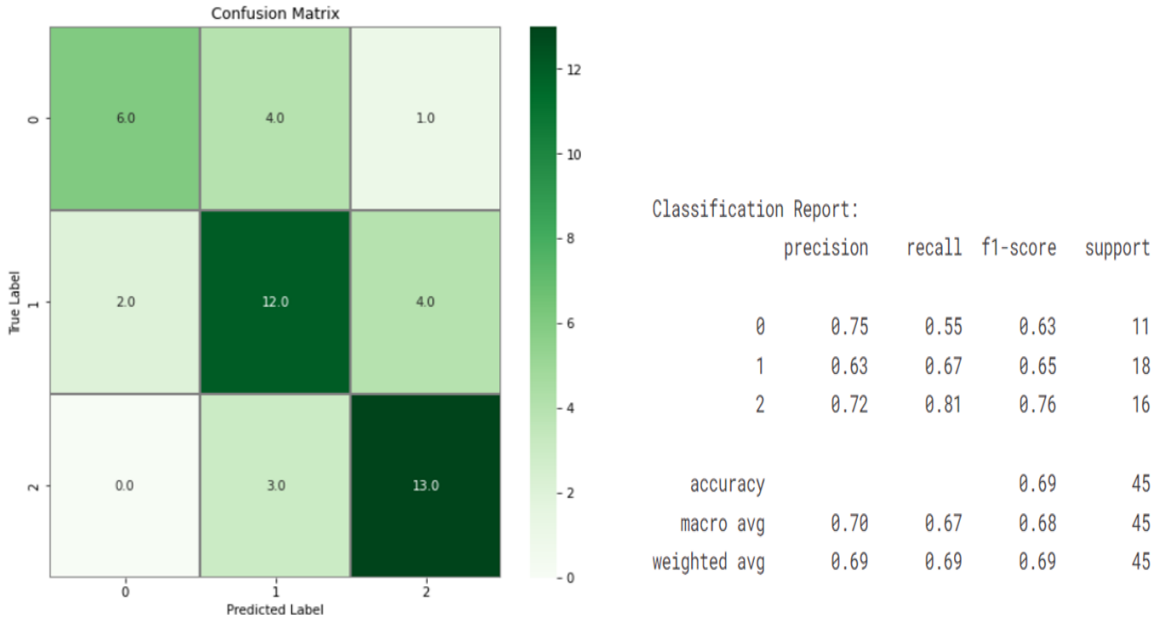


Fig.35 Graphs showing the training and validation accuracy and loss for CNN model



Classification Report:

	precision	recall	f1-score	support
0	0.75	0.55	0.63	11
1	0.63	0.67	0.65	18
2	0.72	0.81	0.76	16
accuracy			0.69	45
macro avg	0.70	0.67	0.68	45
weighted avg	0.69	0.69	0.69	45

Fig.36 Confusion matrix and classification report on test data for CNN model

2.4.2 Performance of Fine-tuned ResNet-50 model

The ResNet-50 has yielded a very good accuracy of 91.59% on the training set and 92.5% on the validation set. The loss is also very less at only 0.245 on the training set. All transfer learning models are only trained for 50 epochs as training for 100 epochs led to overfitting.

From the confusion matrix, we can infer that the hip class is 100 % accurately classified and nearly gable classes are correctly identified except for one. The classification of flat classes is better compared to the previous CNN model and the misclassification of a few flat classes as gable could be due to the slight tilt angle of a few rooftop slopes, making it difficult for the model to classify it to the appropriate class. The overall accuracy on the test samples is 89% with F-1 score averaging to 91%. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. The ROC curve has a value of 1 for hip class as all rooftops are correctly classified.

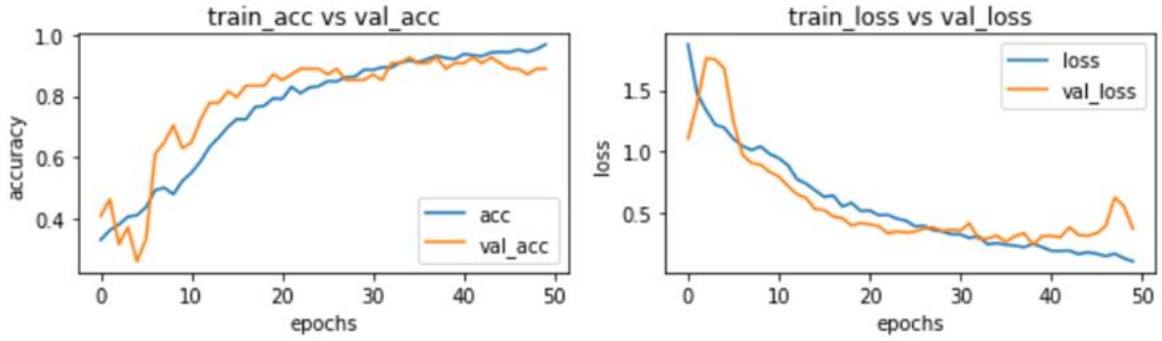


Fig.37 Graphs showing the training and validation accuracy and loss for ResNet-50

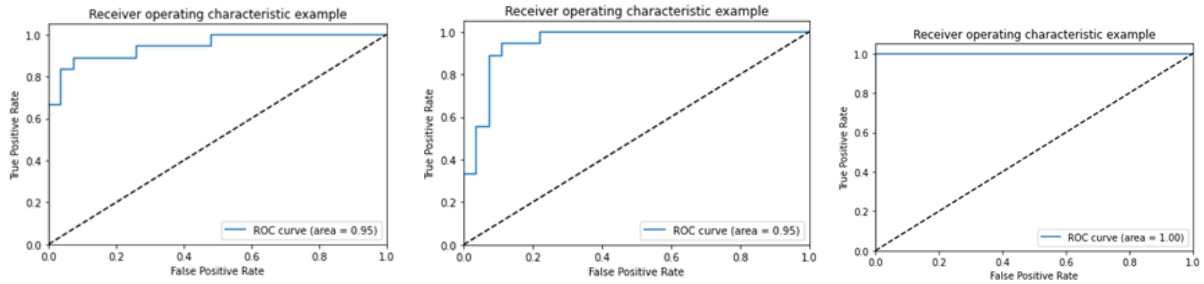
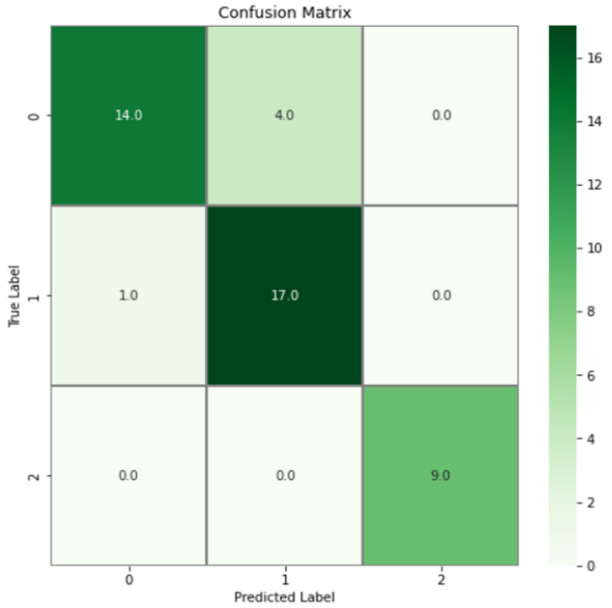


Fig.38 AUC-ROC metrics for ResNet-50 model



Classification Report:

	precision	recall	f1-score	support
0	0.93	0.78	0.85	18
1	0.81	0.94	0.87	18
2	1.00	1.00	1.00	9
accuracy			0.89	45
macro avg	0.91	0.91	0.91	45
weighted avg	0.90	0.89	0.89	45

Fig.39 Confusion matrix and classification report on test data for ResNet-50 model

2.4.3 Performance of Fine-tuned EfficientNetB4 model

The graphs tell us that the accuracy of the model has steadily grown from 40% to 89% and the model loss has significantly reduced from 1.25 to 0.33 at the end of 50 epochs. When compared with the ResNet-50 model which had higher accuracy in correctly classifying hip class, EfficientNetB4 model has achieved higher F-1 score and accuracy in identifying flat roof tops. The average F-1 score, precision, and recall hover around 91%. The AUC-ROC score for flat and gable classes is 0.93 and is a perfect 1.0 for hip.

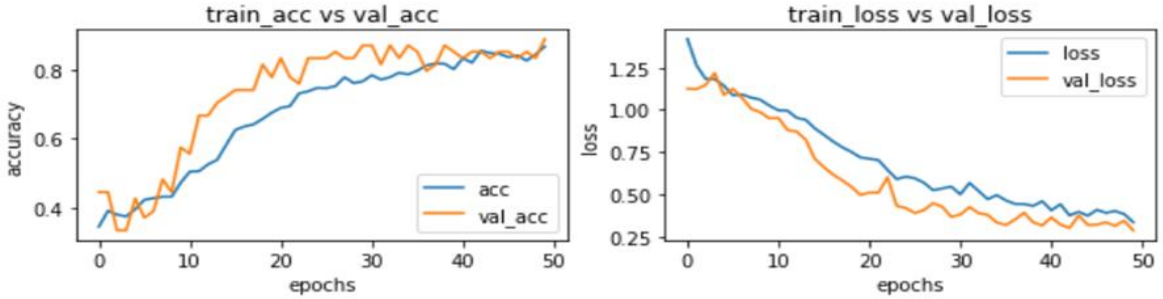


Fig.40 Graphs showing the training and validation accuracy and loss for EfficientNetB4

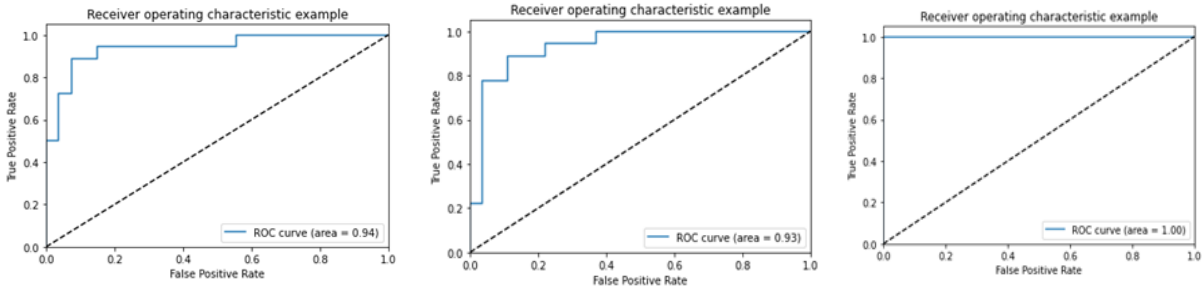
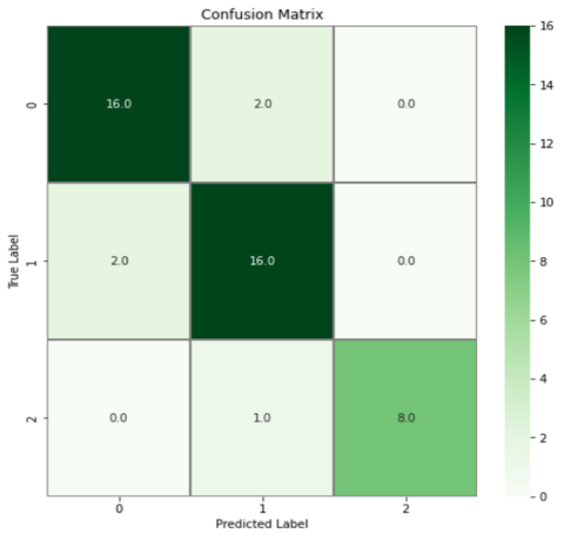


Fig.41 AUC-ROC metrics for EfficientNetB4 model



Classification Report:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	18
1	0.84	0.89	0.86	18
2	1.00	0.89	0.94	9
accuracy			0.89	45
macro avg	0.91	0.89	0.90	45
weighted avg	0.89	0.89	0.89	45

Fig.42 Confusion matrix and classification report on test data for EfficientNetB4 model

2.4.4 Performance of Fine-tuned VGG-16 model

VGG-16 model has achieved the highest accuracy compared to the other three models. The classification accuracy on the validation set is 94.45% while the training accuracy is around 97%. When comparing the loss, there is a gradual decline in training loss over 50 epochs while there are some fluctuations with validation loss. Like other models, this too has classified hip class accurately and the F-1 score for flat class is the highest compared to other models. The overall classification accuracy has totalled to 89%.

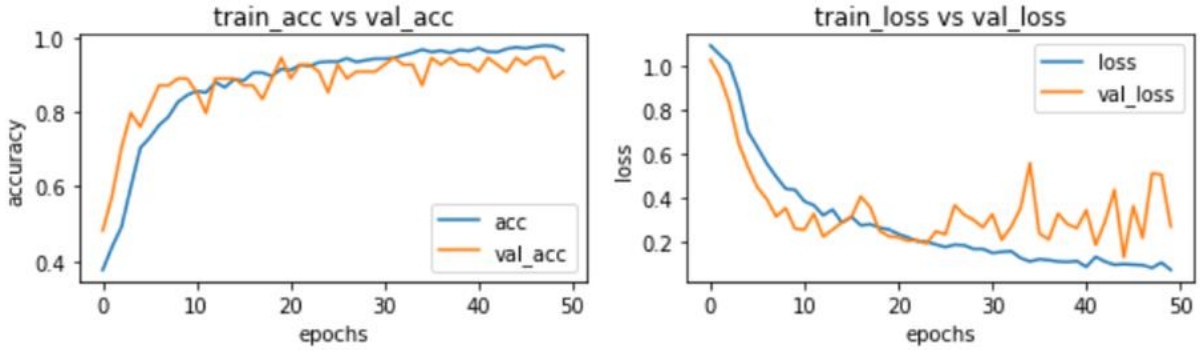


Fig.43 Graphs showing the training and validation accuracy and loss for VGG-16 model

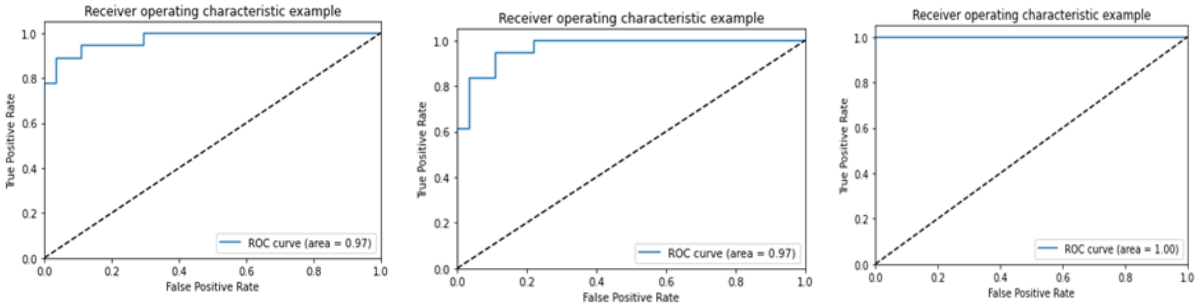
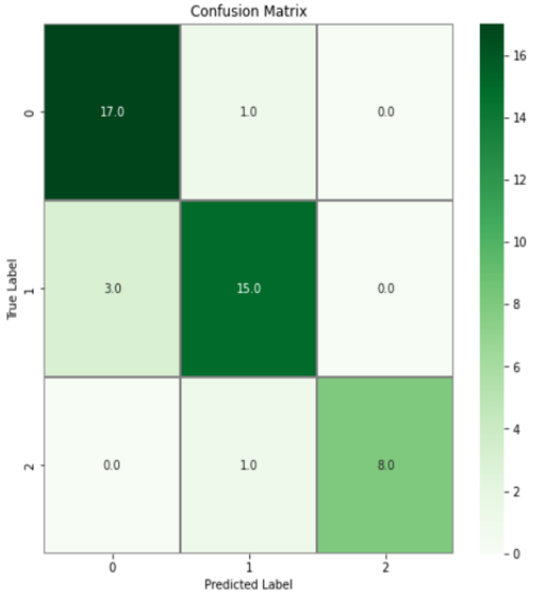


Fig.44 AUC-ROC metrics for VGG-16 model



Classification Report:

	precision	recall	f1-score	support
0	0.85	0.94	0.89	18
1	0.88	0.83	0.86	18
2	1.00	0.89	0.94	9
accuracy			0.89	45
macro avg	0.91	0.89	0.90	45
weighted avg	0.89	0.89	0.89	45

Fig.45 Confusion matrix and classification report on test data for VGG-16 model

Model	Learning rate & Optimizer	Batch Size	Loss	Accuracy (%)	F-1 score (%)
Shallow CNN	0.0001 and RMSProp	16	0.7233-Train 0.8067-Test	69.87-Train 60 - Test	68
ResNet-50	0.00001 and RMSProp	8	0.2450-Train 0.7123-Test	92.59-Train 88.89 - Test	91

Efficient NetB4	0.00001 and Adam	16	0.3359-Train 0.343-Test	86.70-Train 89 - Test	91
VGG16	0.0001 and RMSProp	4	0.2021-Train 0.4894-Test	96.5-Train 88.89 - Test	89

Table 5 Results of different models on our manually labeled dataset

2.5 Validation on Potsdam dataset

As mentioned in section 2.1, we have resorted to manual labeling of roof types. In order to validate our dataset, the trained models were tested on a labeled dataset of roof types from Potsdam in Germany. The dataset contains 2000 samples of images belonging to three classes - Flat, Gable and Hip and we tested our model on 600 randomly picked images. The results are listed in the table below.

Model	Learning rate & Optimizer	Batch Size	Loss	Accuracy (%)	F-1 score
Shallow CNN	0.0001 and RMSProp	16	1.2829	45.67	0.38
ResNet-50	0.00001 and RMSProp	8	0.9248	70.67	0.71
Efficient NetB4	0.00001 and Adam	16	0.7110	65.17	0.66
VGG16	0.0001 and RMSProp	4	0.8507	74	0.73

Table 6 Results of testing our trained models on Potsdam dataset

From the findings, we notice that the accuracy on the Potsdam dataset is hovering around 70% on average when compared to the results on manually labeled dataset. There are several factors which we think could be the reason for the same:

(i) CNN and other transfer learning models were trained on high resolution images whereas the images of the Potsdam dataset are not of very good resolution.

(ii) Nearly all rooftops in Potsdam are brown in color while the colors of rooftops in Christchurch are significantly bright.

With the aforementioned reasons in mind, we can see that an average of 75% accuracy is still a pretty respectable result, especially considering the model was trained on one dataset and evaluated on another.

2.6 Majority Voting

From the above performance results, we notice that transfer learning methods result in higher classification accuracy when compared to our shallow CNN model. However in these transfer learning models, we see each model performs classification well on only some parts. Hence an ensembling approach is used as it can make better predictions and achieve better performance than any single contributing model. Ensembling also reduces the spread or dispersion of the predictions and model performance. The ensembling method applied here is majority voting.

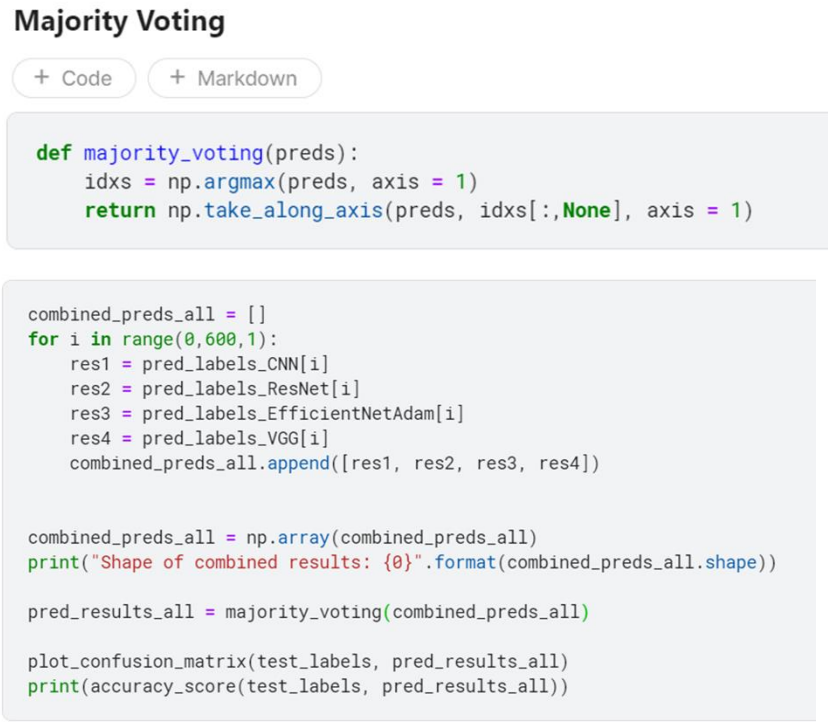


Fig.46 Code implementation of majority voting

From the previous discussion of performance of different models, we find out that VGG16 was efficient in identifying hip roof types. Similarly, ResNet very well

classified flat roofs from other rooftop types and EfficientNet was able to classify a large chunk of gable roof types correctly. Majority voting is thus incorporated to further boost the performance.

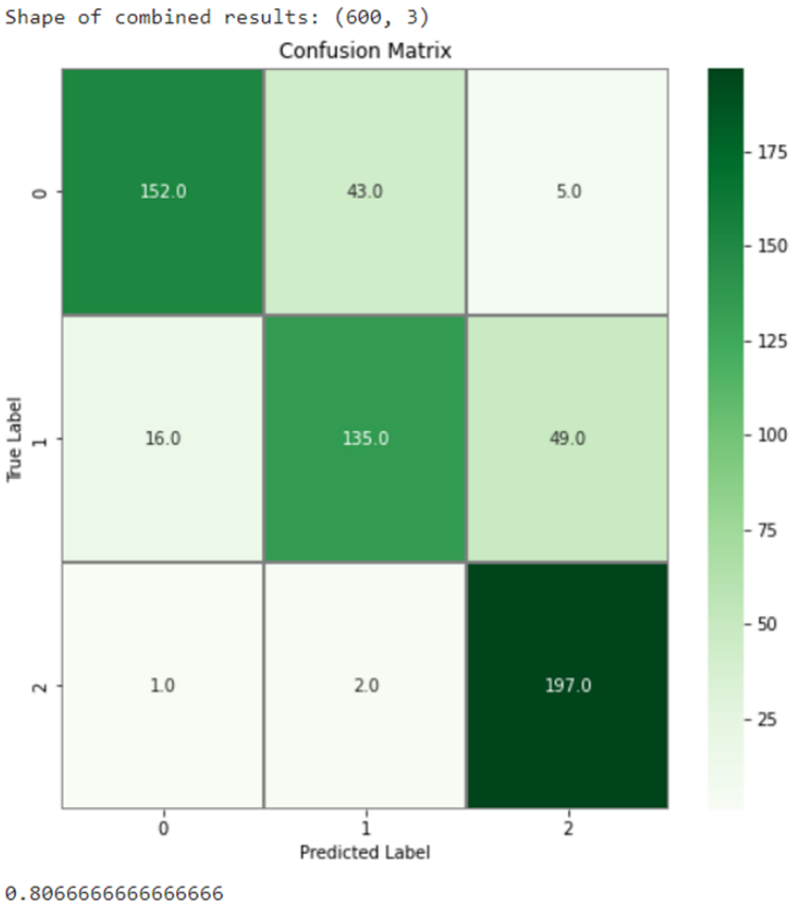


Fig.47 Results on Potsdam dataset after applying majority voting

After applying majority voting, the accuracy has increased by 5.67% and the overall classification accuracy is now 80.67%.

2.7 Boundary detection

2.7.1 White Balancing

White balance is the initial step. This step must be completed first as it is color dependent. White balance is used to remove any coloured haze from an image so that it seems to be under white light. Edge recognition methods can be hampered by haze on an image since it reduces local contrast, which can lead to edges being ignored or incorrectly recognised. This is a major issue with satellite images as they are taken with varying incident angles for the light, or may be exacerbated by weather and atmospheric variances. Due to the highly variable lighting conditions under which satellite imagery is gathered, this is an extremely impactful step on the resulting quality of edge detection.

1. **White patch algorithm**

```
#White balance image using White patch algorithm
def white_patch(image, percentile=100):
    white_patch_image = img_as_ubyte((image*1.0 /
                                      np.percentile(image,percentile,
                                      axis=(0, 1))).clip(0, 1))
    return white_patch_image
```

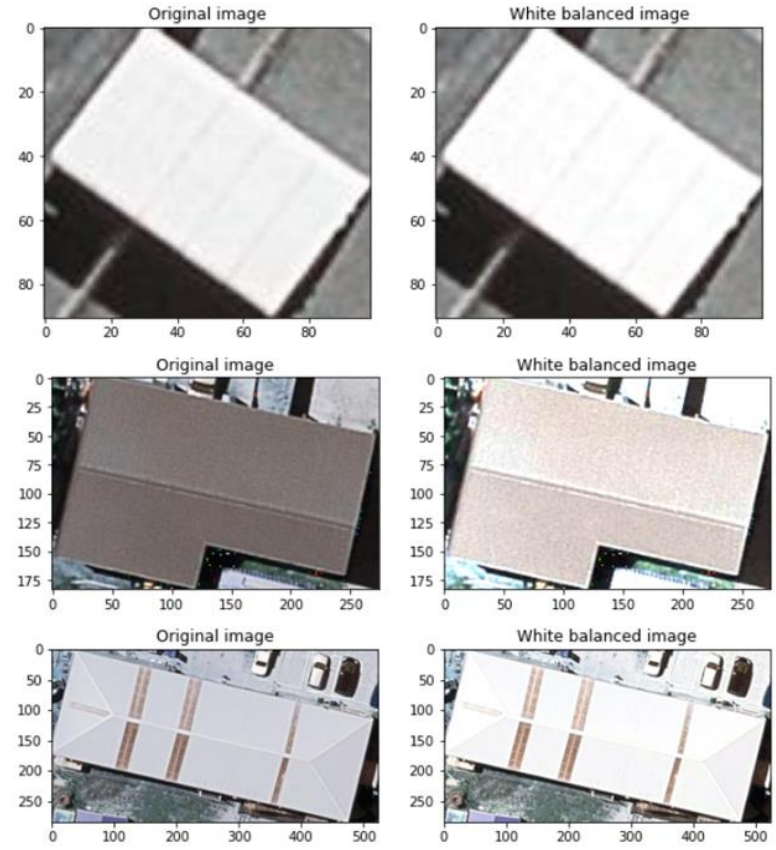


Fig.48 Images after performing white balancing to remove haze

2.7.2 Gaussian Blur

Gaussian blurring is a smoothing technique that reduces various types of noise or accentuates edge contrast within the image. Here, a kernel size of 3×3 is used with $\sigma = 1.5$

2.7.3 Auto Canny Edge Detection

Canny edge detection algorithm is a popular edge detection algorithm that is multi-staged that includes noise reduction, finding intensity gradients and hysteresis suppression. After white patching and converting the image to grayscale, Gaussian blur is applied. This is followed by auto canny edge detection. The median of single channel pixel intensity is calculated. The upper and lower threshold is calculated by using mean and variance of an image intensity. Laplacian operator is a derivative operator which is used to find edges in an image. It is a second order derivative mask. Here, the Laplacian operator is also applied on the gray image and then edges are detected. From fig.49, we can infer that applying auto canny after Gaussian blur provides better results comparatively.

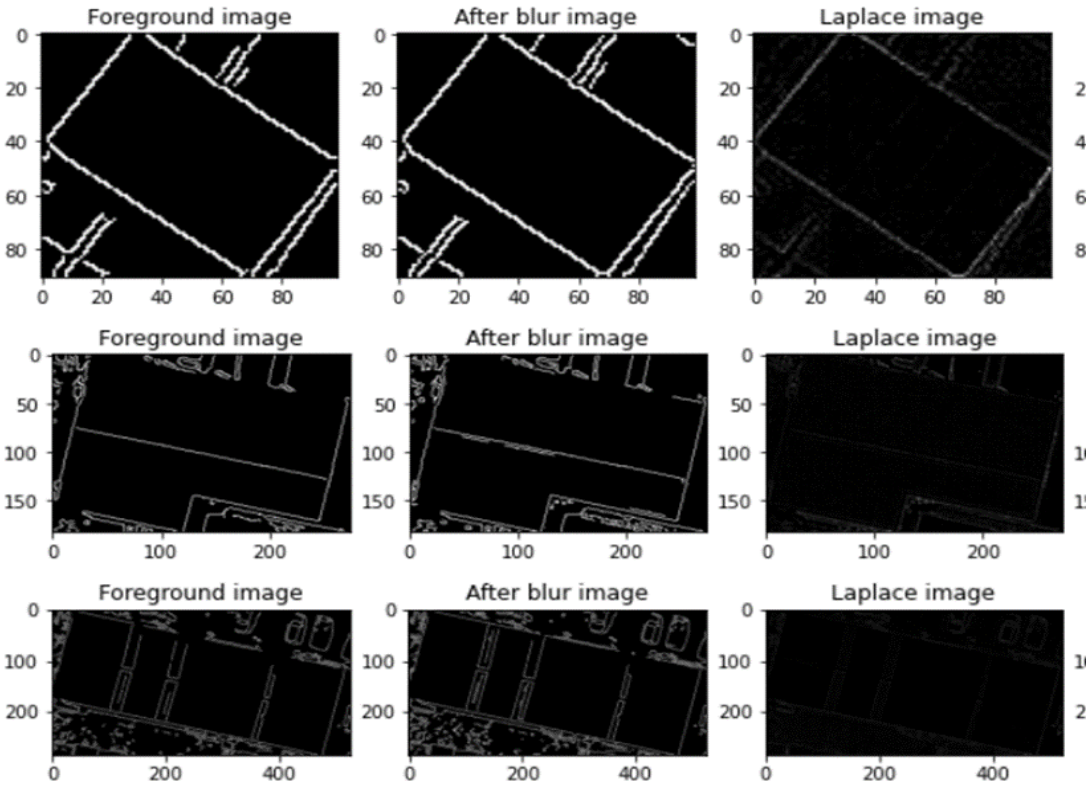


Fig.49 Boundaries detected on rooftops by applying different techniques

METRICS FOR EVALUATION

For building detection segmentation:

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

To quantify the accuracy of our model for building detection, we use Jaccard coefficient which is 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.

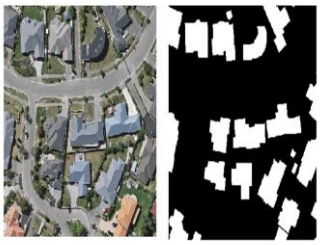
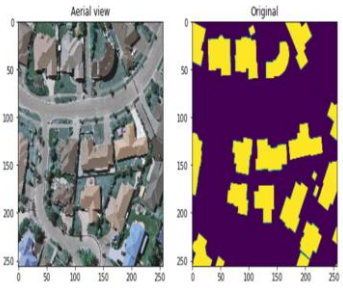
$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \qquad Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

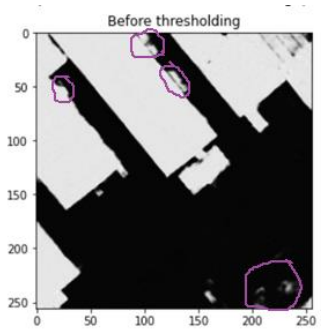
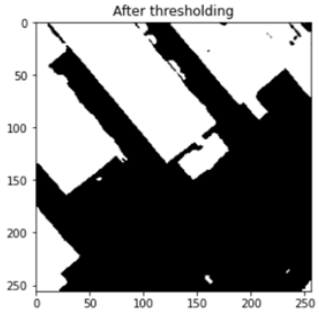
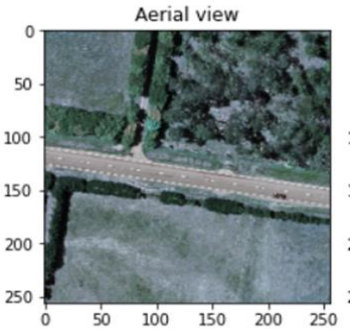
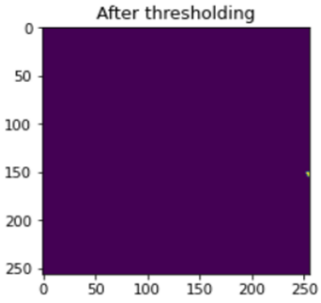
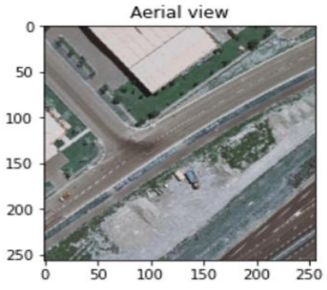
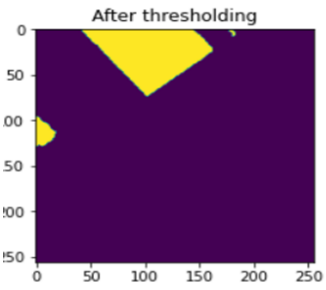
$$F1 = 2.\frac{Precision \times Recall}{Precision + Recall}$$

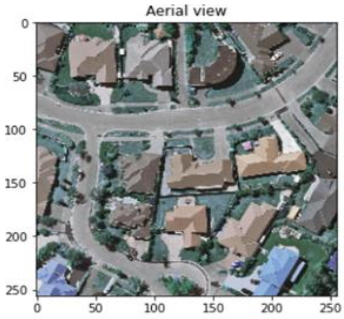
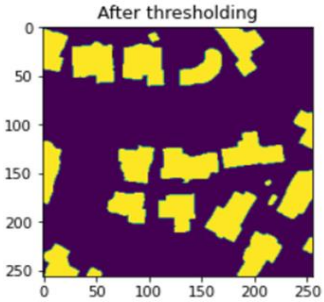
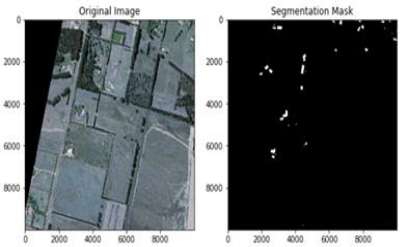
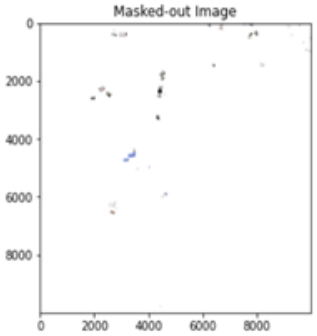
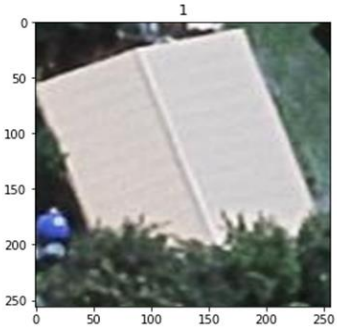
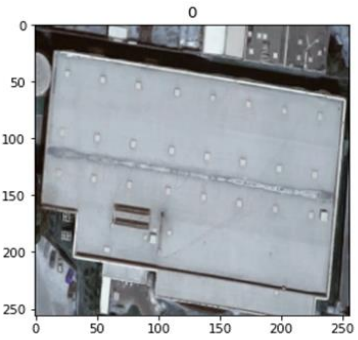
❑ **AUC-ROC**

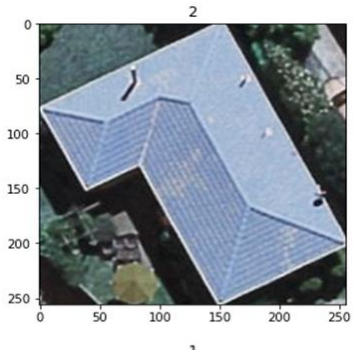
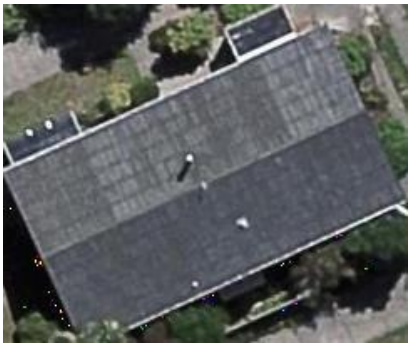
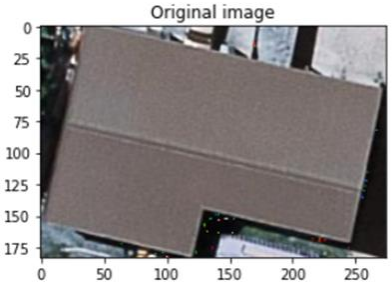
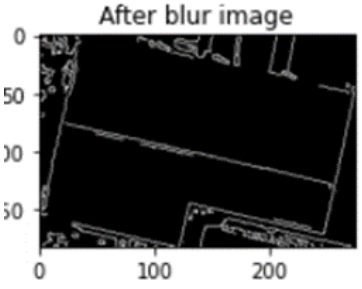
The Receiver Operator Characteristic (ROC) is a probability curve that plots the TPR(True Positive Rate) against the FPR(False Positive Rate) at various threshold values and separates the ‘signal’ from the ‘noise’. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish 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 images.		
TC_02	Clipping of ground truth masks.		
TC_03	Normalization		

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

TC_07	Building segmentation of various buildings in a single image.		
TC_08	Rooftop extraction by background subtraction		
TC_09	Classification of Gable image		ResNet50 - 1 (Gable) Shallow CNN - 1 (Gable) EfficientNetB4 - 1 (Gable) VGG16 - 1 (Gable)
TC_10	Classification of flat image		ResNet50 - 0 (Flat) Shallow CNN - 1 (Gable) EfficientNetB4 - 0 (Flat) VGG16 - 0 (Flat)

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_11	Classification of hip image		ResNet50 - 2 (Hip) Shallow CNN - 2 (Hip) EfficientNetB4 - 2 (Hip) VGG16 - 2 (Hip)
TC_12	Majority Voting (Unequal predictions)		CNN Result: 2 ResNet Result: 1 EfficientNetB4-Adam Result: 1 VGG Result: 1 [[2 1 1 1]] [0, 3, 1] 1 Image christchurch_173_343.jpg belongs to class: 1 3 models predict as Gable while 1 model predicts as Flat
TC_13	Majority Voting (Equal predictions)		CNN Result: 2 ResNet Result: 2 EfficientNetB4-Adam Result: 2 VGG Result: 2 [[2 2 2 2]] [0, 0, 4] 2 Image christchurch_173_395.jpg belongs to class: 2 All models predict as class hip
TC_14	Auto Canny Edge detection		

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.