FINAL YEAR PROJECT 2021-2022
TEAM 18

Building extraction and roof type classification of aerial images for maximal PV panel installation

PROJECT GUIDE
Prof Dr. P. Uma Maheswari

TEAM MEMBERS
Shruthi M - 2018103592
Gayathri M - 2018103535
Jayapriya M - 2018103029

INTRODUCTION

- Climate change has become a global concern and harnessing renewable resources is the way to construct a sustainable environment. Potential tapping of solar power for generating electricity has gained enormous popularity and people are increasingly gravitating toward the PV revolution.
- However, traditional approaches, such as online assessment of rooftops are time-consuming and expensive. By automating the process of building roof extraction for PV panel placement, a lot of money and time can be saved.
- 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 of Christchurch 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.
- Building segmentation is widely utilized in urban planning, topography mapping, disaster assessment, analyzing geographical land occupation.
- The first stage is building detection from aerial satellite images. We propose a deep learning framework called MultiRes UNet, with ResPath skip connections between the encoder and decoder structure. Better the segmentation of buildings, maximum is the solar potential of each of the rooftop areas.
- This is followed by rooftop classification from the extracted buildings with different SOTA models as roof classification is used in new building design, retrofitting existing roofs, and efficient solar integration on building rooftops.
- Finally, based on the type of roofs, we determine the maximum number of PV panels by applying a maximum fitting approach.

OVERALL OBJECTIVES

To maximize the placement of photovoltaic panels on the rooftop for an aerial satellite image, the following steps are to be performed:

- Detect buildings in a given satellite image using MultiRes U-Net model and perform background subtraction to extract the building rooftops alone.
- Label the extracted building rooftops into different classes - Flat, Complex, Gable, Hip to create a dataset and train pre-trained models for roof type classification.
- Perform edge detection on the extracted rooftops to mark boundaries to find the area for PV panel installation.
- Use module fitting algorithm to find the maximum no of solar panels based on type of roof to maximize energy consumption.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
1.	**An aerial image segmentation approach based on enhanced multi-scale convolutional neural network, 2019** Xiang Li, Yuchen Jiang, Hu Peng and Shen Yin in 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)	1. Segmentation model is performed using an encoder-decoder architecture. 2. 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. 3. Add an auxiliary loss function after the cascaded dilated convolution	1. From the aspect of design and training, the approach does not involve manual features and does not require specific preprocessing or post-processing, which can reduce the influence of subjective factors 2. The auxiliary loss function helps to make the network converge faster and optimize.	1. Segmentation of large buildings work well but boundaries and middle parts are misaligned. 2. The bulges on the boundaries are lost and edges are not detected properly. 3. The algorithm performs well in one of the subset (countryside and forest) but does not perform well when tested on a different subset.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
2.	**Convolutional Neural Network Based Solar Photovoltaic Panel Detection in Satellite Photos, 2017** Vladimir Golovko, Sergei Bezobrazov, Alexander Kroshchanka and Anatoliy Sachenko in 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications	1. Collect data from Google Maps by giving the latitude and longitude details and store them in geojson format. 2. Perform pre-processing techniques like image resizing, image sharpening. 3. Train a 6 layer CNN model.	1. Here, the authors have used the low-quality satellite imagery (Google Maps photos), instead of the high resolution color satellite orthoimagery that enables decreasing the requirements for the approach. 2. Simple 6 layer CNN model.	1. Simple CNN model hasn't led to efficient segmentation of solar panels. 2. Bad quality satellite images have led to inaccurate classification. 3. No validation on the dataset as in some cases solar panels look similar to roof tops.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
3.	**Deep Convolutional Neural Network Application on Rooftop Detection for Aerial Imagery, 2019** Mengge Chen, Jonathan Li, in Journal of Computational Vision and Imaging Systems	1. It is primarily based on Mask R-CNN with 3 stages. 2. Feature extraction is based on existing deep learning model. 3. RPN (Regional Proposal Network) is used to find RoI. 4. Object classification is then performed.	1. Efficient and feasible approach to extract detached house from aerial images. 2. RoIAlign method is used instead of RoIPool for better feature extraction.	1. Edges of the building are not detected properly. 2. Training data was less and hence less accuray. 3. Comparatively less precision with other new state of art models.
4.	**Solar Potential Analysis Of Rooftops Using Satellite Imagery, 2019** Akash Kumar, Delhi Technology University, in *ArXiv* abs/1812.11606	1. Dataset is manually collected for India. 2. Adaptive Edge Detection and Contours are focused to segment out rooftop boundaries and obstacles present inside them along with polygon shape approximation.	1.Provides a comparative analysis of the solar potential of the building. 2. Several types of the rooftop are considered to learn the intra-class variations.	1. The image quality of satellite imagery is very deficient hence the edges are not detected properly. 2. There are some outliers that are plotting solar panels outside the building rooftop area.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
5.	**Deep learning based roof type classification using very high resolution aerial imagery, 2021** M. Buyukdemircioglu , R. Can , S. Kocaman in The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLIII-B3-2021 XXIV ISPRS Congress	1. Using UltraCam Falcon large-format digital camera orthophotos with 10cm spatial resolution is captured and roofs are manually classified into 6 different labels. 2. Data augmentation is applied and a shallow CNN architecture is trained. 3. The prediction is investigated by comparing with three different pre-trained CNN models, i.e. VGG-16, EfficientNetB4, and ResNet-50.	1. Simple CNN model are hence easier to implement. 2. Requires nominal hardware specifications. 3. The shallow CNN model has achieved 80% accuracy.	1. Since the roof images were clipped automatically from the orthophotos, there are few buildings with overlap. 2. Half-hip roofs are not classified properly and F1 score obtained for them is very low. 3. Different hyperparameter tuning was not done for the shallow CNN architecture.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
6.	**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** Bodhiswatta Chatterjee, Charalambos Poullis in 2019 16th Conference on Computer and Robot Vision (CRV)	1. ICTNet: a novel network with the underlying architecture of a fully convolutional network, infused with feature re-calibrated Dense blocks at each layer. 2. It is combined with dense blocks, and Squeeze-and-Excitation (SE) blocks. 3. Reconstruction is done by extruding the extracted boundaries of the buildings and comparative analysis is made between the two.	1. Has addressed the task of using few parameters to process large chunks of data. 2. With no 3D information on the buildings, the authors have used the building boundaries as a proxy for the reconstruction process. 3. Has got better overall IoU compared to other methods.	1. There is no loss function for the reconstruction accuracy. 2. There is high discrepancy on per-building IoU due to the fact that ground truth images used for training contain errors and are manually created. 3. Reconstruction accuracy is consistently lower than classification accuracy by an average of 4.43% ± 1.65%. 4. Need extensive hardware specifications to train the model.

LITERATURE SURVEY

S.NO	CITATION	METHODOLOGY	ADVANTAGES	LIMITATION
7.	**Understanding rooftop PV panel semantic segmentation of satellite and aerial images for better using machine learning, 2021** Peiran Li , Haoran Zhang , Zhiling Guo, in Advances in Applied Energy, Volume 4, 100057, ISSN 2666-7924	1. Data pre-processing involves collecting patch satellite images from Google for the city of Heilbron and manually labelling them. 2. Object proportion distribution in image-level and object occurrence possibility at pixel level is statistically analysed. 3. SOTA PV segmentation model (DeepSolar) is used to extract visual features. 4. Local Binary Pattern (LBP) is used for texture feature extraction & color histograms for color feature extraction.	1. Class imbalance of PV and non-PV panels in rooftops is resolved by hard sampling, soft sampling. 2. The homogenous textural feature using LBP has served as an additional part for some easily confused cases to improve the robustness of the model.	1. IOU is less than the acceptable range (0.5) for 1.2m resolution images. 2. Patch overlapping occurs while stitching the tiles and this leads to segmentation errors. 3. Lighting conditions resulted in different clustering groups in color clustering of PV/Non-PV and has led to misclassification.

SUMMARY OF ISSUES IN LITERATURE SURVEY

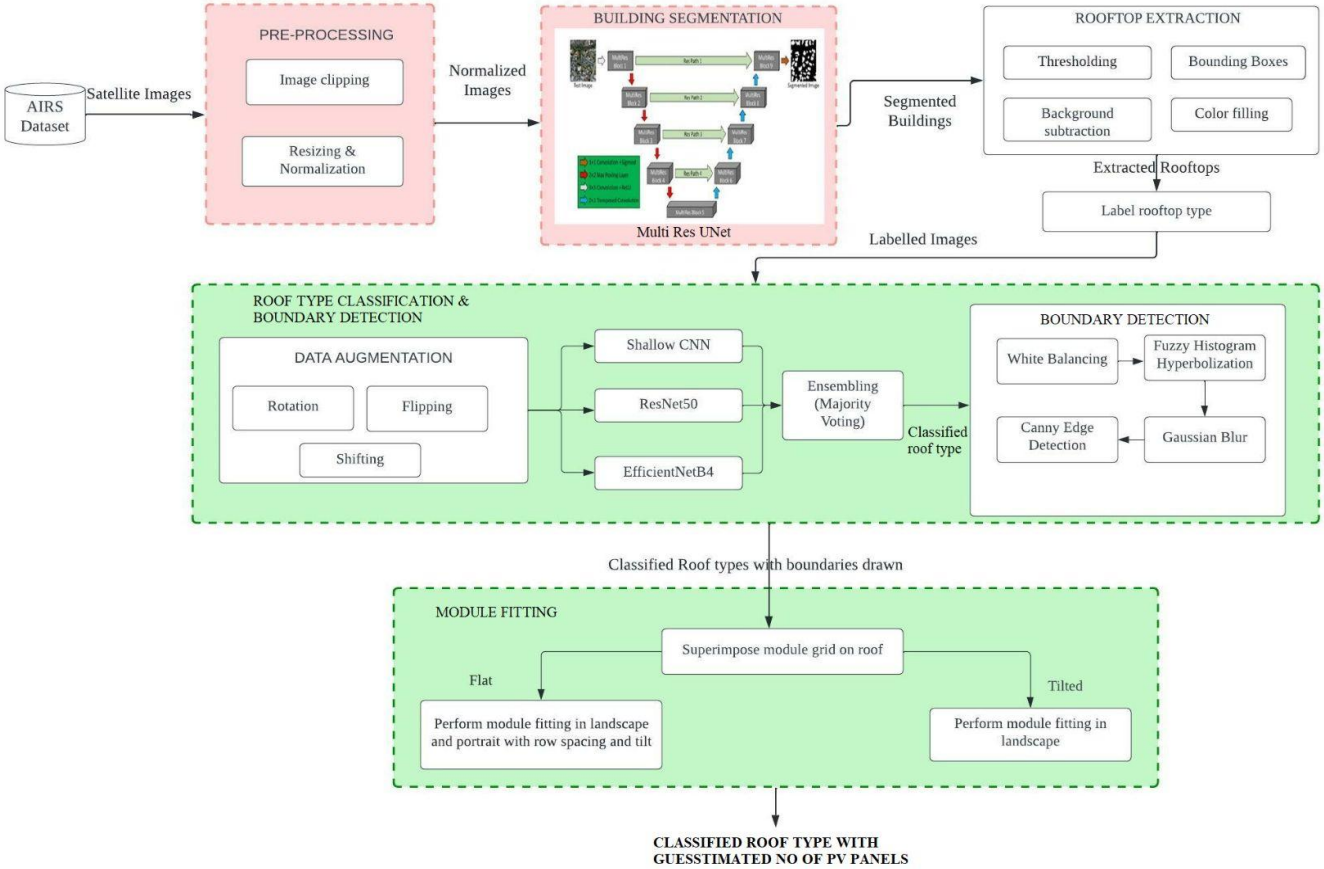
The U-Net model with auxiliary loss function proposed in [1] has aided in network convergence, however the segmentation of middle parts in buildings are misaligned and the bulges on the boundaries are lost. One major issue is that the algorithm performs well only in one subset (countryside and forest) but not in another. The authors of [2] have employed a CNN based solar PV panel detection but this did not result in efficient segmentation of solar panels. Because some solar panels resemble roof tops, poor quality satellite pictures taken from Google Maps have led to erroneous classification and there is no validation on the dataset. In [3], the Mask-RCNN method is used for feature extraction and is able to efficiently extract detached houses in aerial imagery. However, building edges are not effectively demarcated, as a result of the short training dataset, which has resulted in low accuracy and precision compared to other SOTA models. [4] uses a combination of image processing techniques, including Adaptive Edge Detection and contours, to segment out rooftop boundaries. 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. An investigative analysis is made with the shallow CNN model developed by authors of [5] and other pre-trained models like VGG16, ResNet50. As the roof images were clipped automatically from orthophotos, there are few buildings with overlap. Half-hip roofs are not classified properly and 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. ICTNet, a novel framework developed in [6], leverages border localization for classification and reconstruction of buildings. 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. Training the model requires extensive hardware specifications and high RAM. In [7], SOTA model (DeepSolar) is used for segmentation along with LBP (Local Binary Pattern) for texture feature extraction. The major drawback is that lighting conditions caused distinct colour clustering groups in PV/Non-PV colour clustering, resulting in misclassification along with IOU being less than the acceptable range (0.5) for 1.2m resolution images.

PROPOSED SYSTEM

Our proposed system aims to do the following:

- ★ Use SOTA MultiRes UNet to perform building segmentation and extraction on AIRS dataset.
- ★ Train different classifier models to classify the different type of roofs.
- ★ Mark boundaries on rooftops using edge detection algorithms.
- ★ Guesstimate the no of panels that can be fitted based on the type of roof.

OVERALL ARCHITECTURE



1st Review Comments

- ★ Validation for roof types of AIRS dataset for classification.
- ★ Validation for 3rd module (PV module fitting).
- ★ Analyse time taken to run all models for classification of roof types.

OVERALL ARCHITECTURE

The above block diagram gives a high level overview on the 3 modules.

To begin, our proposed system includes two pre-processing steps. 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 to extract the rooftops.

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

The final module resorts to providing a guesstimate of the number of PV panels that can be fitted in the given rooftop which is achieved by the maximum fitting algorithm.

LIST OF MODULES

MODULE 1: BUILDING DETECTION

MODULE II: ROOF TYPE CLASSIFICATION AND BOUNDARY DETECTION

MODULE III: MAXIMAL FITTING ALGORITHM

MODULE DESIGN

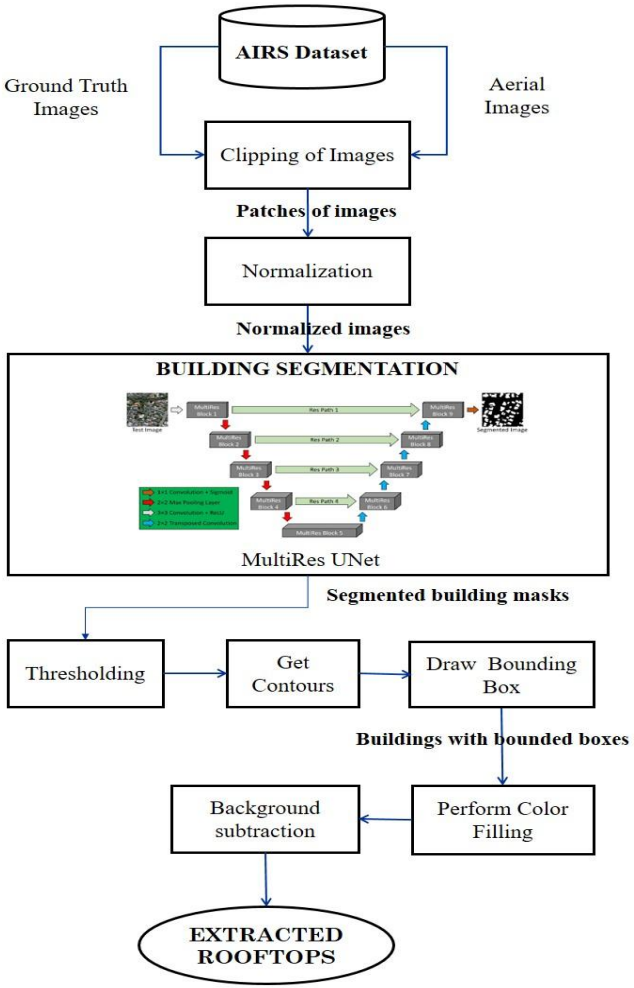
MODULE I: BUILDING DETECTION

INPUT: AIRS Dataset
OUTPUT: Rooftops of different buildings

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

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

- The final step involves background subtraction.
  Input: Binary mask of buildings.
  Output: Rooftops of different buildings.



PSEUDO-CODE

Pre-Processing

Clipping:

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

Normalization:

1. Resize := cv2.resize(img, (256,256), cv2.INTER_CUBIC).
2. Normalization :=
X_std = (X - X.min(axis=0)) / (X.max - X.min).
X_scaled = X_std * (max - min) + min.

Training the MultiRes UNet Architecture

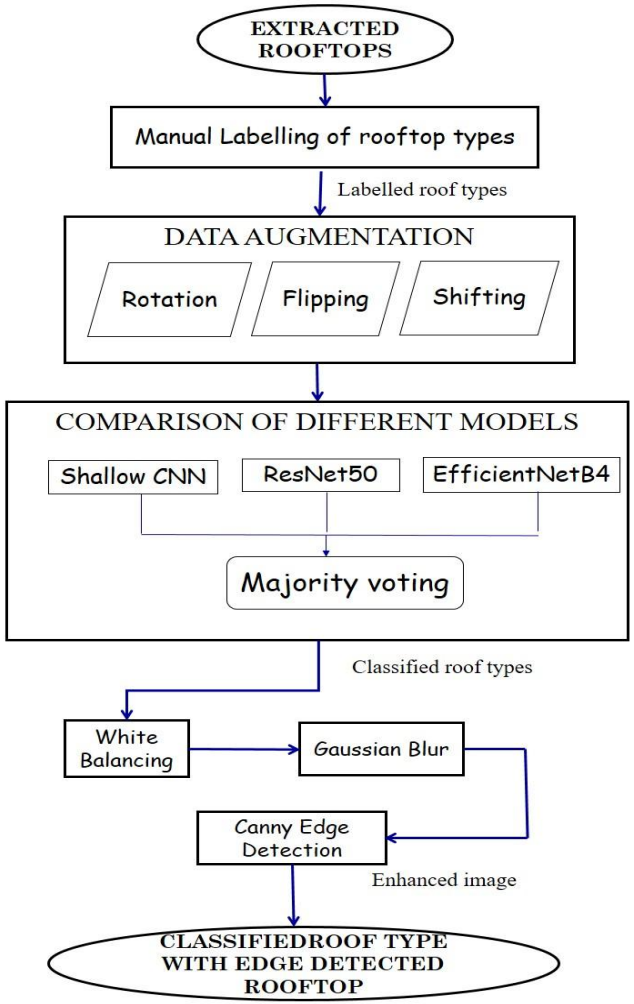
1. Split dataset into train - 1548 images, validation - 36 images and testing - 144 images.
2. Define the MultiRes model.
3. do:
    3.1 Tweak hyperparameters: Set learning_rate := 0.0001, batch_size := 8, epochs := 100, optimizer := Adam, loss := binary cross entropy
    3.2 Train the model
    3.3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

Background Subtraction

Get contours & draw bounding boxes:

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

MODULE DESIGN



MODULE II: ROOF TYPE CLASSIFICATION & BOUNDARY DETECTION

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

- The first step is **manually labeling** the rooftops into 3 different classes - Flat, Gable, Hip.
- This is followed by **data augmentation** with rotation, flipping and shifting as transformations.
- Following this, we try 4 different models - customized CNN, ResNet50, EfficientNetB4 and VGG16 perform **majority voting**
  Input: Rooftops with labels.
  Output: Classified rooftop type.
- The final step involves **boundary detection** performed by White Balancing, Gaussian blur and Canny edge detection.
  Input: Classified rooftop type.
  Output: Boundary detected rooftops.

PSEUDO-CODE

Data Augmentation

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

Comparison of different models

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

Boundary Detection

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

2. Gaussian Blur - Try for different kernel size and σ.

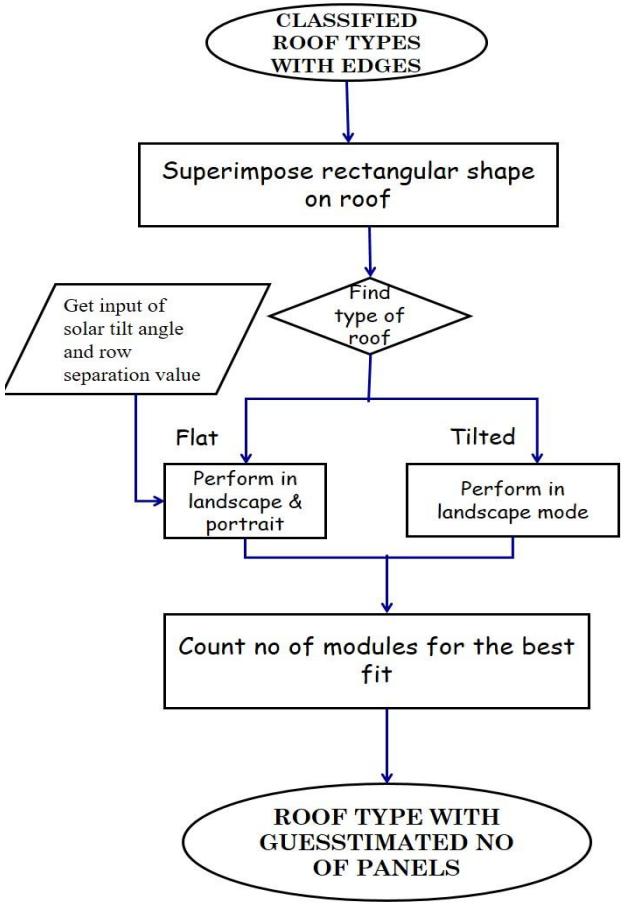
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

MODULE DESIGN

MODULE III: PV MODULE FITTING

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

- The boundary detected rooftops are superimposed with rectangular PV module shape.
- Based on the type of roof:
  - 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.



PSEUDO-CODE

Calculating the PV panels

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

IMPLEMENTATION

DATASET

AIRS dataset covering the full area of Christchurch in New Zealand with satellite images and ground truth masks.

Training set:	857 images
Validation Set:	90 images
Testing Set:	90 images
Spatial dimensions:	10000 ╳ 10000 pixels
Spatial resolution:	7.5 cm

EXPERIMENTAL SETUP

- ❏ Tensorflow backend.
- ❏ Keras Framework.
- ❏ Environment used: Colab Pro with 16BG, T4 GPU and Kaggle GPU notebooks
- ❏ Frontend integrated with flask.

MODULE I - BUILDING DETECTION

1.1 Clipping original aerial images

- Direct segmentation of large satellite images is difficult.
- Dimensions of 10000 * 10000 pixels is cut into 36 smaller patches of size 1536 * 1536 by **sliding window** technique.

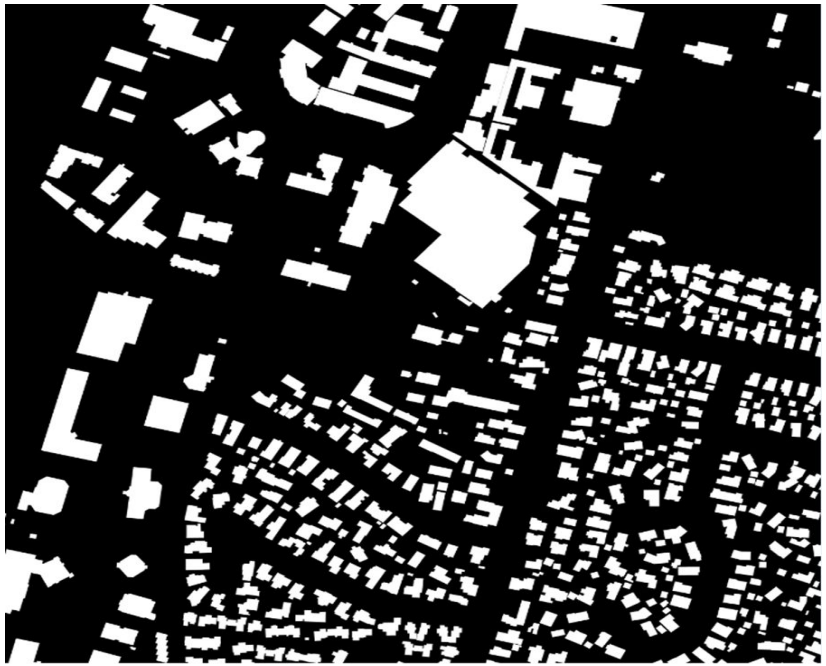
Aerial satellite image

Patches of image after clipping

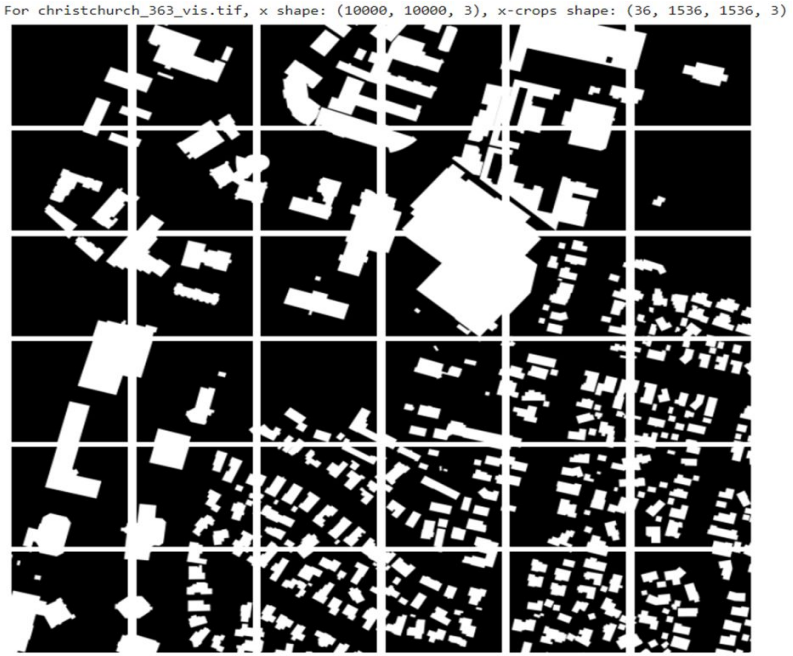
MODULE I - BUILDING DETECTION

1.2 Clipping corresponding mask images

➢ Corresponding ground truth masks for satellite images also needs to be clipped.



Ground truth segmented image



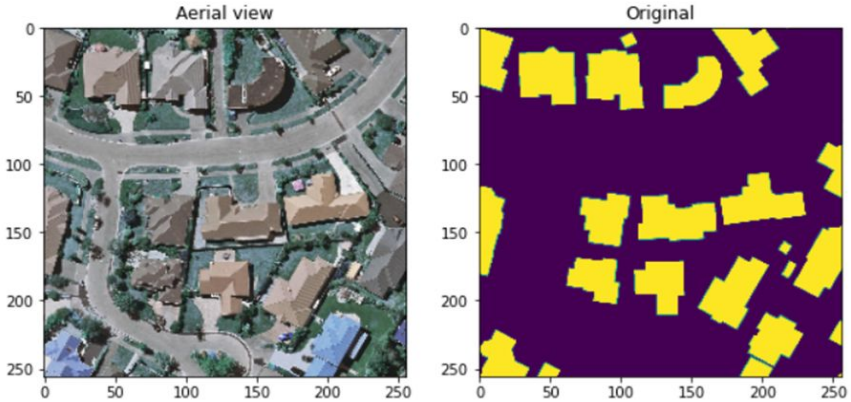
Ground truth patches after clipping

1.3 Resizing and Normalization

As images of size 1536 * 1536 are still huge to process and train on complex neural nets, we resize the images to 256 * 256 with INTER_CUBIC interpolation method as it leads to better resolution of images.

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

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

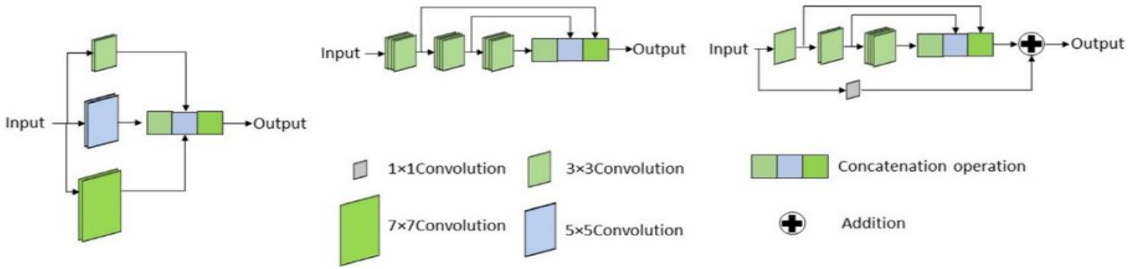


Aerial view and ground truth mask after resizing and employing min-max scaler.

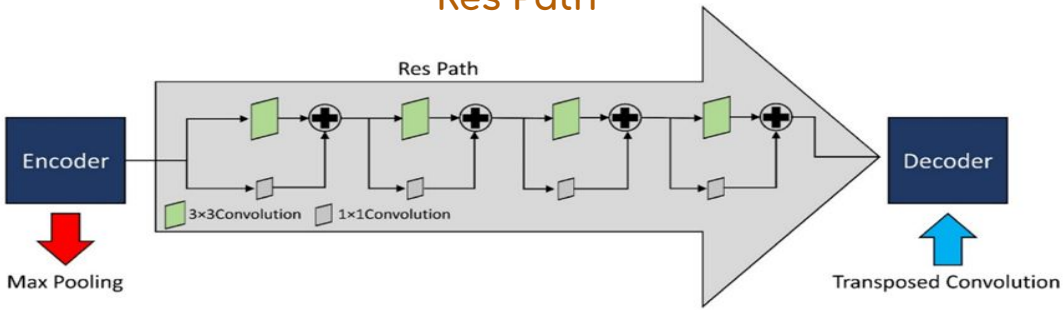
1.4 Adapting the MultiRes UNet architecture

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

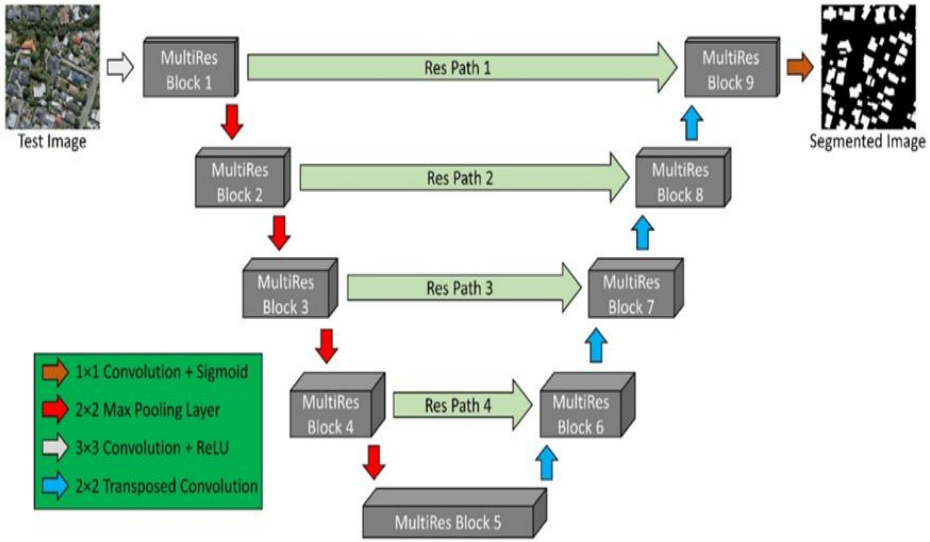
MultiRes Block



Res Path



Architecture of MultiRes UNet



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

Code Snapshots

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

    W = alpha * U

    shortcut = inp

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

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

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

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

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

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

    return out
```

MultiRes Block

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

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

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

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

    for i in range(length-1):

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

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

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

    return out
```

Res Path

Code Snapshots

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

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

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

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

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

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

    mresblock5 = MultiResBlock(32*16, pool4)

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

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

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

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

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

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

    return MultiResModel
```

Brief Model Summary

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

```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
conv2d_1 (Conv2D)	(None, 256, 256, 8)	216	['input_1[0][0]']
batch_normalization_1 (BatchNo rmalization)	(None, 256, 256, 8)	24	['conv2d_1[0][0]']
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 rmalization)	(None, 256, 256, 17)	51	['conv2d_2[0][0]']
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 rmalization)	(None, 256, 256, 26)	78	['conv2d_3[0][0]']
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 ormalization)	(None, 256, 256, 51)	153	['conv2d_52[0][0]']
batch_normalization_82 (BatchN ormalization)	(None, 256, 256, 51)	204	['concatenate_12[0][0]']
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 ormalization)	(None, 256, 256, 51)	204	['activation_55[0][0]']
conv2d_56 (Conv2D)	(None, 256, 256, 1)	51	['batch_normalization_83[0][0]']
batch_normalization_84 (BatchN ormalization)	(None, 256, 256, 1)	3	['conv2d_56[0][0]']
activation_56 (Activation)	(None, 256, 256, 1)	0	['batch_normalization_84[0][0]']

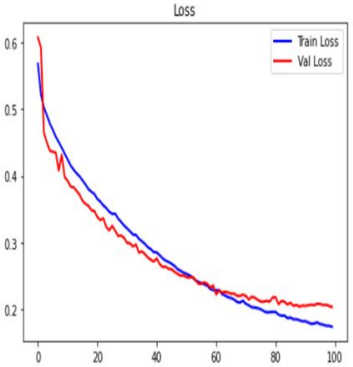
```
Total params: 9,906,494
Trainable params: 9,876,980
Non-trainable params: 29,514
```

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

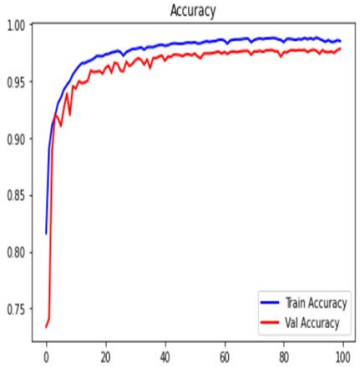
```
Epoch 1/100
194/194 [==============================] - 213s 1s/step - loss: 0.5683 - IOU: 0.4220 - mcc: 0.5200 - dice_coef: 0.5787 - accuracy: 0.8161 - val_loss: 0.6078 - val_IOU: 5.4112e-05 - v
Epoch 2/100
194/194 [==============================] - 198s 1s/step - loss: 0.5220 - IOU: 0.5455 - mcc: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

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

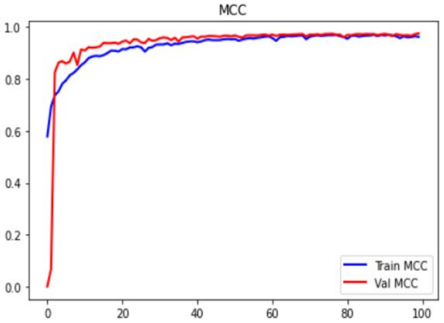
Model Loss

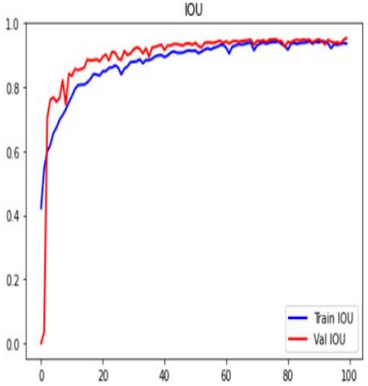


Model Accuracy

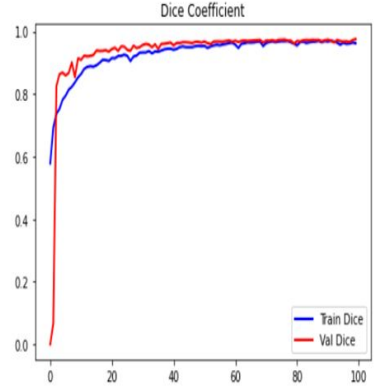


Model MCC





Model IOU

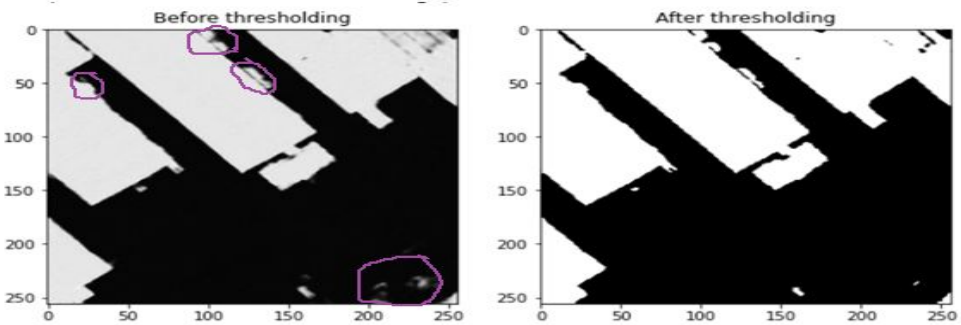


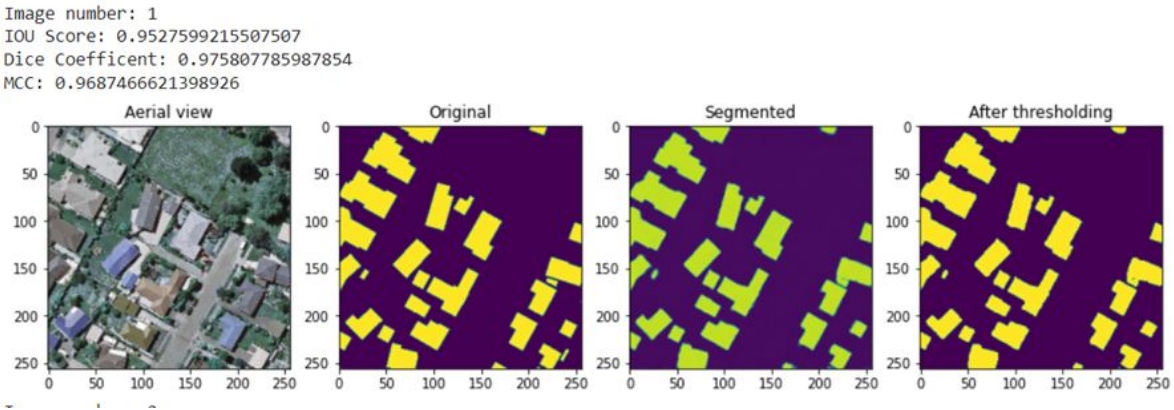
Model Dice Coefficient

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

1.5 Applying threshold

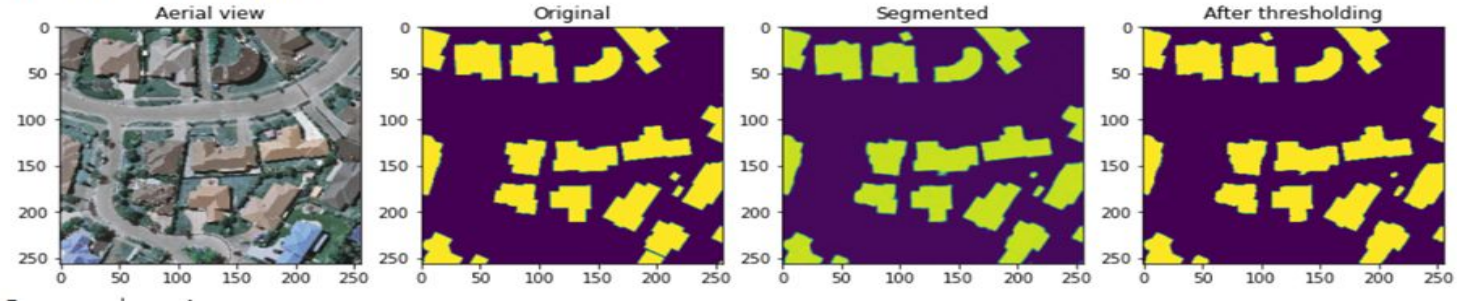
Threshold is set to 0.5 to delineate the boundaries of buildings and differentiate the edges properly. The violet color markings show the difference before and after the threshold is applied.





Results of building segmentation with MultiRes UNet model

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




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

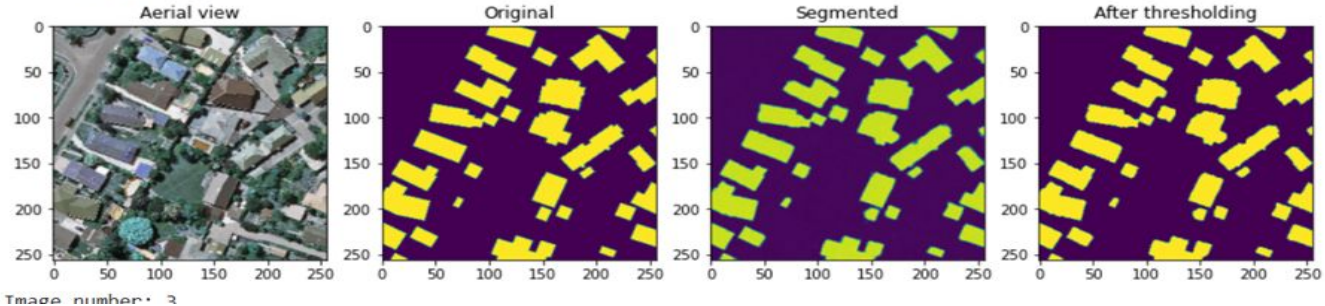
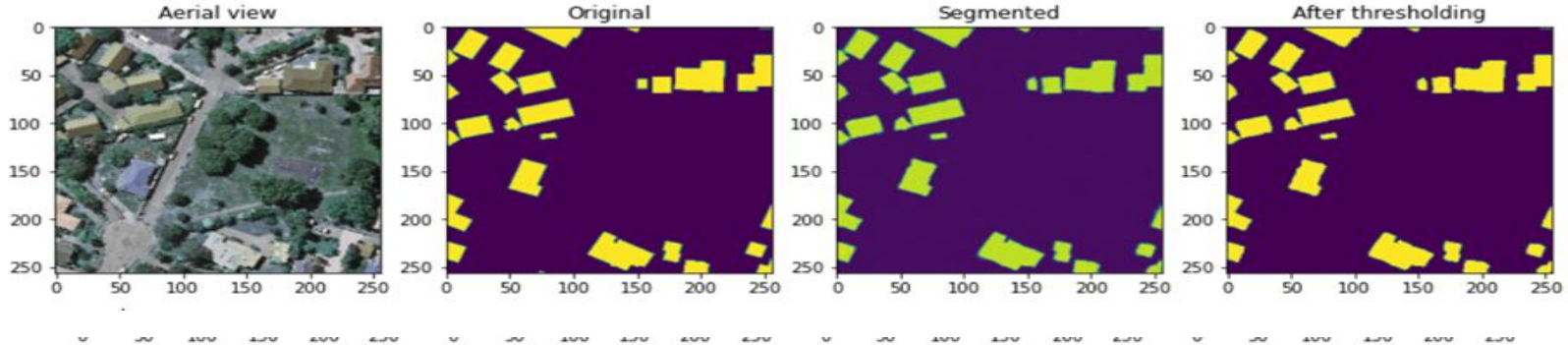


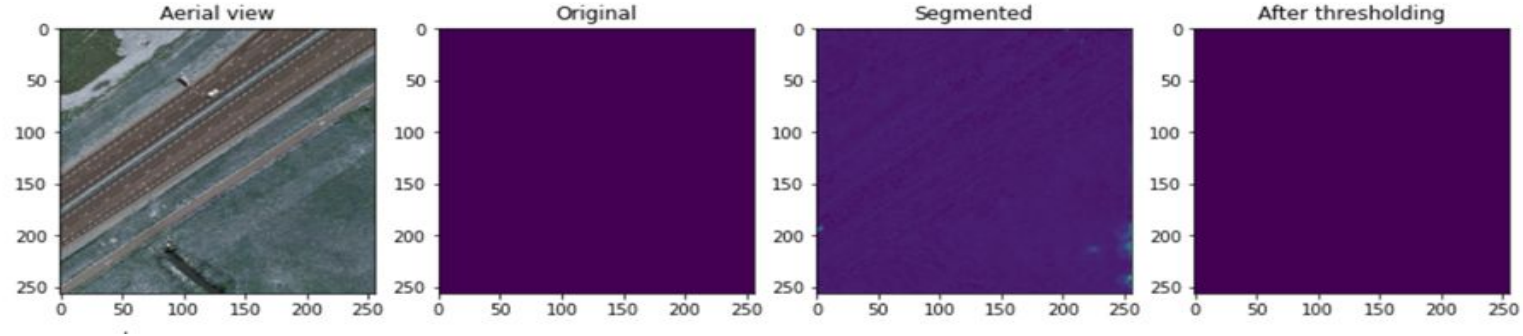
Image number: 3

Results of building segmentation with MultiRes UNet model

```
Image number: 3
IOU Score: 0.9486061930656433
Dice Coefficent: 0.9736239910125732
MCC: 0.9692211151123047
```




```
Image number: 137
IOU Score: 1.0
Dice Coefficent: 1.0
MCC: 1.0
```



Results of building segmentation with MultiRes UNet model

MODULE I - BUILDING DETECTION

1.6 Rooftop Extraction

1.6.1 Get contours

- Contours are used to **detect the borders** of objects and **localize** them easily in an image.
- **Grayscale** transformation and then **binary thresholding** is applied on it following which the contours of buildings are identified.

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 = []
```

Code snapshots of finding contours

MODULE I - BUILDING DETECTION

1.6 Rooftop Extraction

1.6.2 Drawing bounding box

- Bounding boxes are used to highlight the **regions 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)
```

Bounding boxes drawn around segmented mask images.

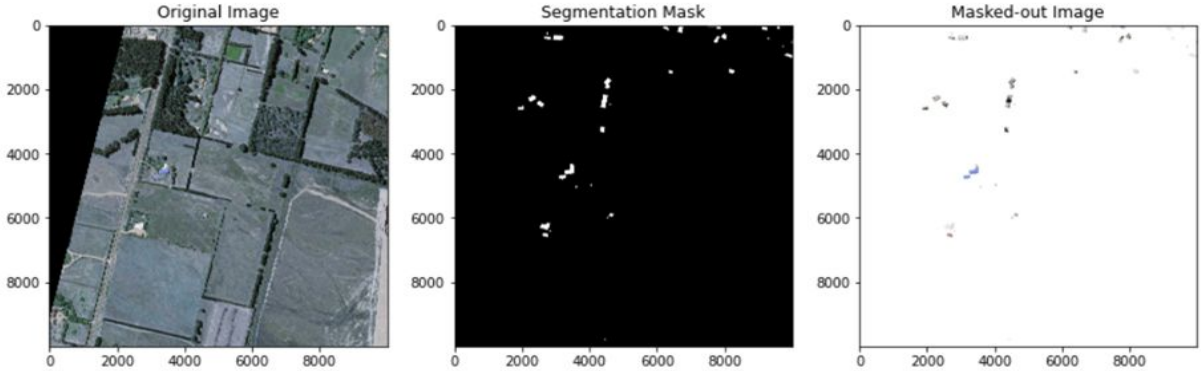
1.6 Rooftop Extraction

1.6.3 Background Subtraction

- Overlay segmentation mask on top of aerial satellite image.
- Subtract bounded box segmented regions detected in earlier step with satellite image.

Background Subtraction

```
# Post-process
new_seg_image_gray = cv2.cvtColor(seg_with_bounded_box, cv2.COLOR_RGB2GRAY) # Convert the mask to grayscale
img_np = np.asarray(img) # Convert the PIL image to a numpy array
masked_out = cv2.bitwise_and(img_np, img_np, mask=new_seg_image_gray) # Blend the mask
masked_out_new = np.where(masked_out != 0, masked_out, 255) # Remove the background
```



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

MODULE II - ROOF TYPE CLASSIFICATION AND BOUNDARY DETECTION

2.1 Preparation of dataset and labeling roof type

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 manually labelled the data into three classes.

Samples of images of each class from Christchurch, New Zealand

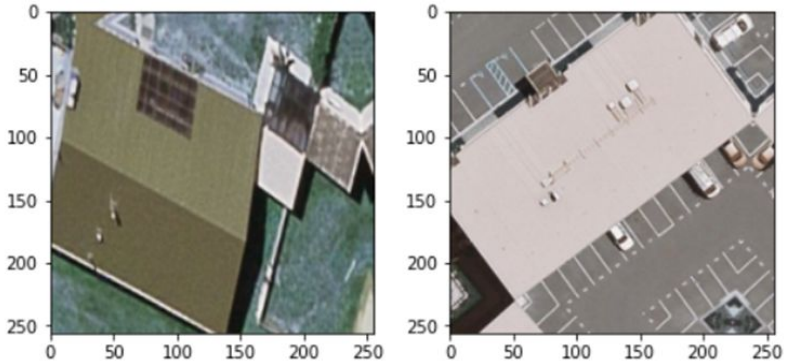
FLAT		
GABLE		
HIP		

2.2 Data Augmentation

We have employed data augmentation approaches to increase the dataset size for better model training. Rotation, shifting and flipping are the three main techniques used here.

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)
```



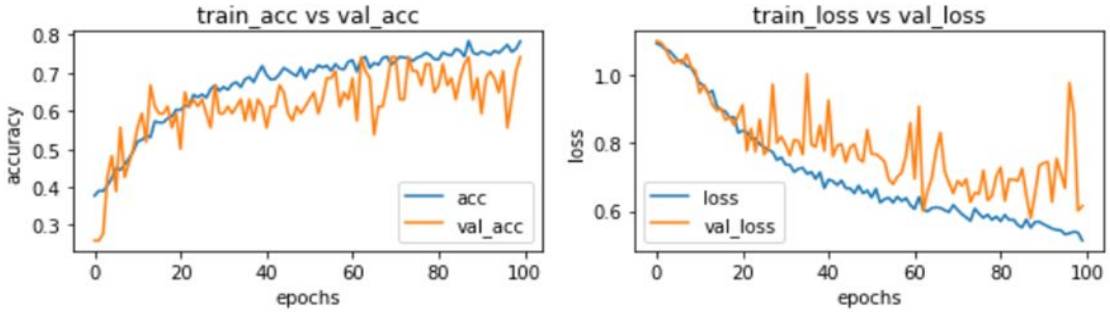
Results of image after performing data augmentation

2.3 Roof type classification

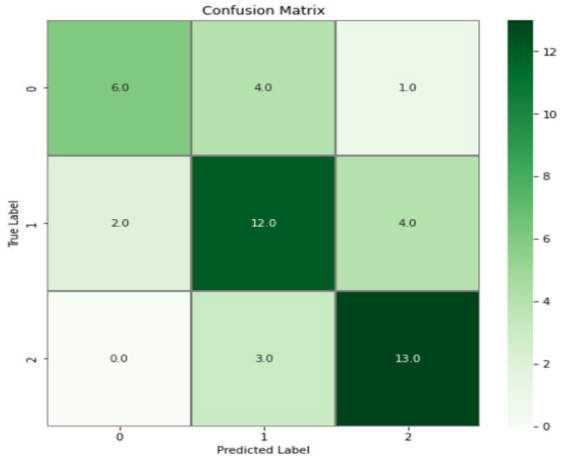
2.3.1 Shallow CNN Model

```
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
```

Model summary of CNN model



Graphs showing the training and validation accuracy and loss for CNN model



Classification Report:

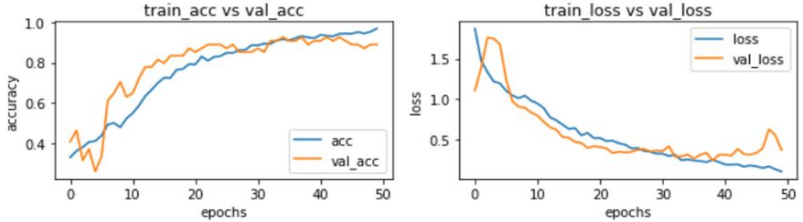
	precision	recall	f1-score	support
0	0.75	0.55	0.63	11
1	0.63	0.67	0.65	18
2	0.72	0.81	0.76	16
accuracy			0.69	45
macro avg	0.70	0.67	0.68	45
weighted avg	0.69	0.69	0.69	45

Confusion matrix and classification report on test data for CNN model

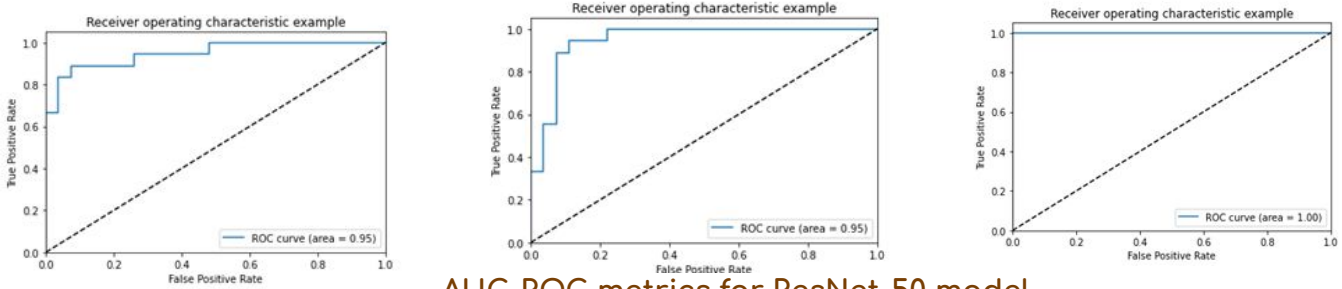
2.3.2 ResNet50

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

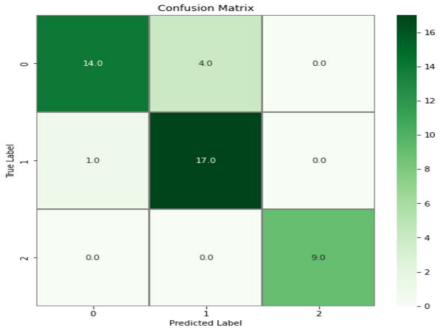
Architecture of ResNet50 model



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



AUC-ROC metrics for ResNet-50 model

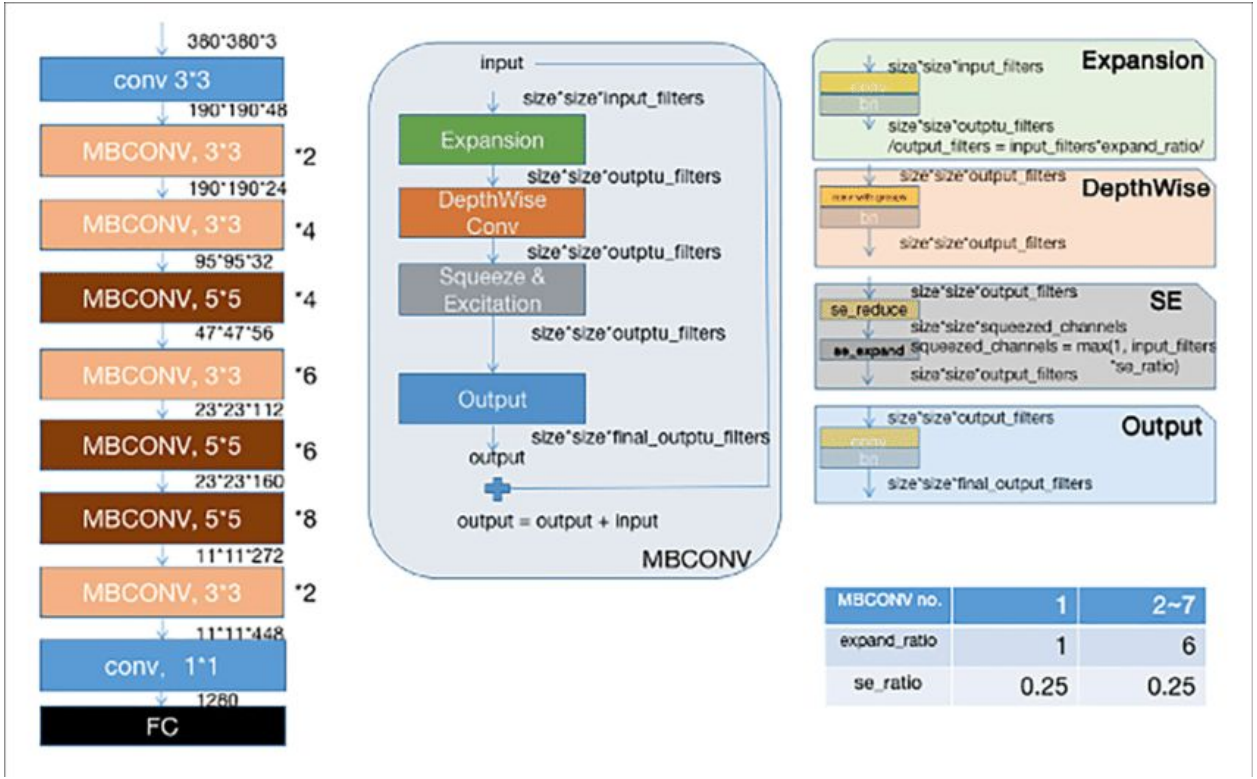


Classification Report:

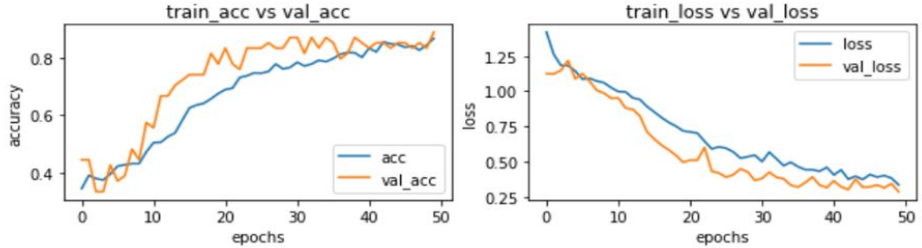
	precision	recall	f1-score	support
0	0.93	0.78	0.85	18
1	0.81	0.94	0.87	18
2	1.00	1.00	1.00	9
accuracy			0.89	45
macro avg	0.91	0.91	0.91	45
weighted avg	0.90	0.89	0.89	45

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

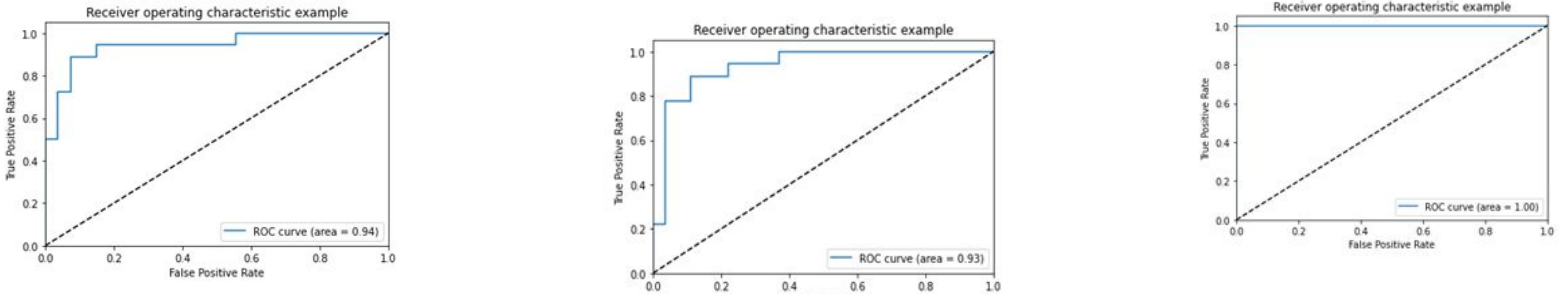
2.3.3 EfficientNetB4



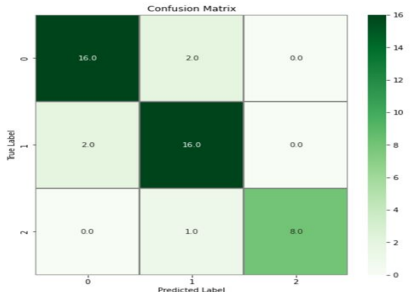
Architecture of EfficientNetB4 model



Graphs showing the training and validation accuracy and loss for EfficientNetB4



AUC-ROC metrics for EfficientNetB4 model

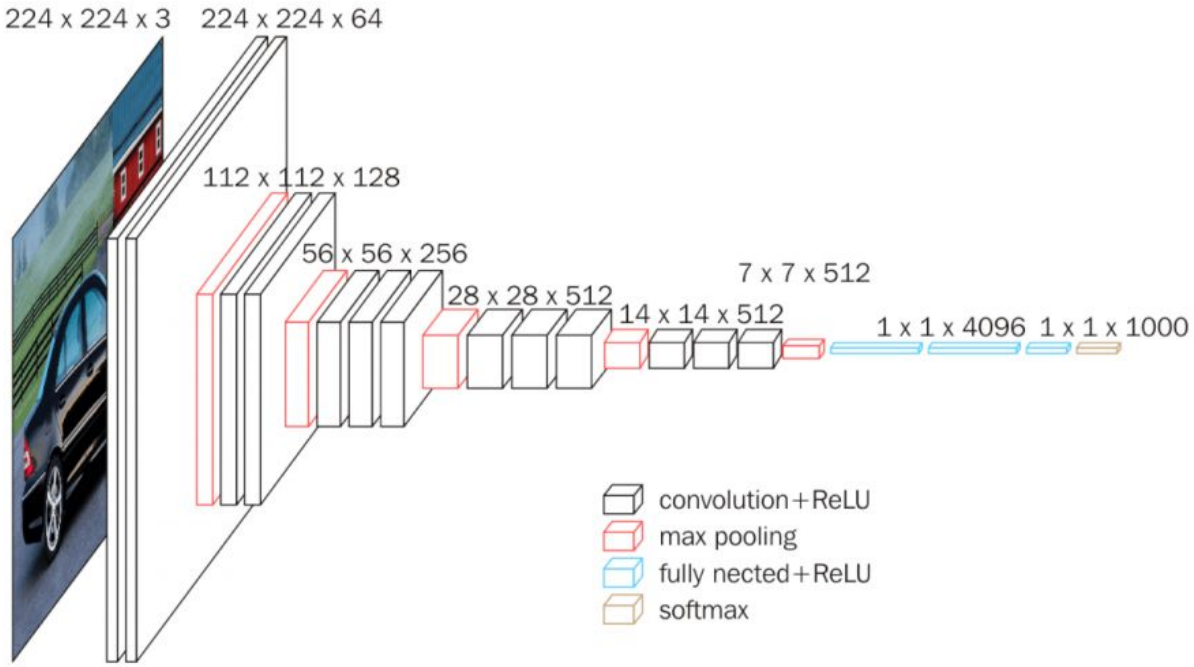


Classification Report:

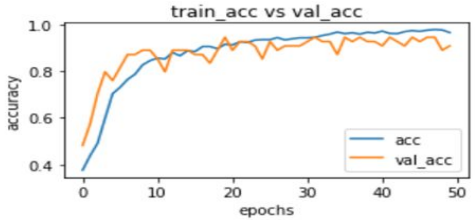
	precision	recall	f1-score	support
0	0.89	0.89	0.89	18
1	0.84	0.89	0.86	18
2	1.00	0.89	0.94	9
accuracy			0.89	45
macro avg	0.91	0.89	0.90	45
weighted avg	0.89	0.89	0.89	45

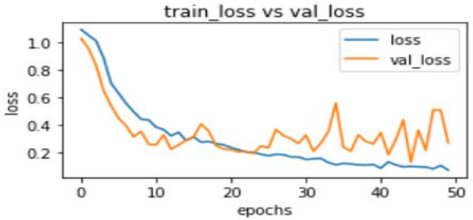
Confusion matrix and classification report on test data for EfficientNetB4 model

2.3.4 VGG16

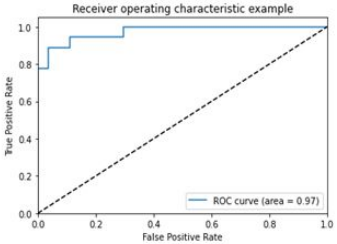


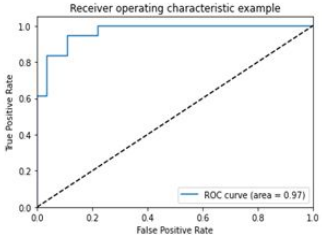
Architecture of VGG16 model

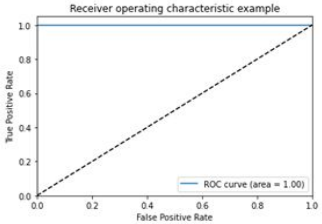




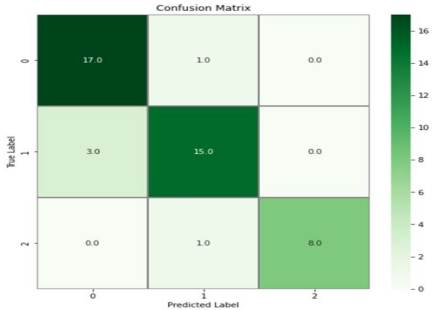
Graphs showing the training and validation accuracy and loss for VGG-16 model







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

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

COMPARISON OF RESULTS WITH DIFFERENT MODELS

Model	Learning rate & Optimizer	Batch size	Loss	Accuracy (%)	F-1 score
Shallow CNN	0.0001 & RMSProp	16	0.7233 - Train 0.8067 - Test	69.87 - Train 60 - Test	0.69
ResNet- 50	0.00001 & RMSProp	8	0.2450 - Train 0.7123 - Test	92.59 - Train 88.89 - Test	0.91
EfficientNetB4	0.00001 & Adam	16	0.3359 - Train 0.343 - Test	86.70 - Train 89 - Test	0.91
VGG16	0.0001 & RMSProp	4	0.2021 - Train 0.4894 - Test	96.5 - Train 88.89 - Test	0.89

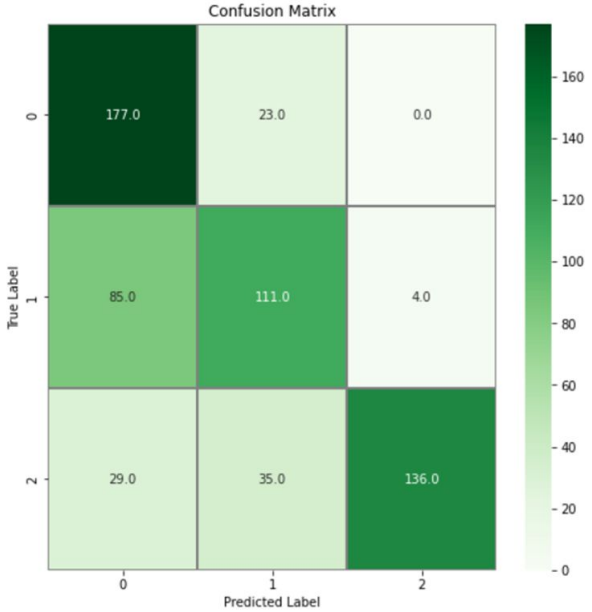
Results of different models on our manually labeled dataset

VALIDATION OF MODELS ON POTSDAM DATASET

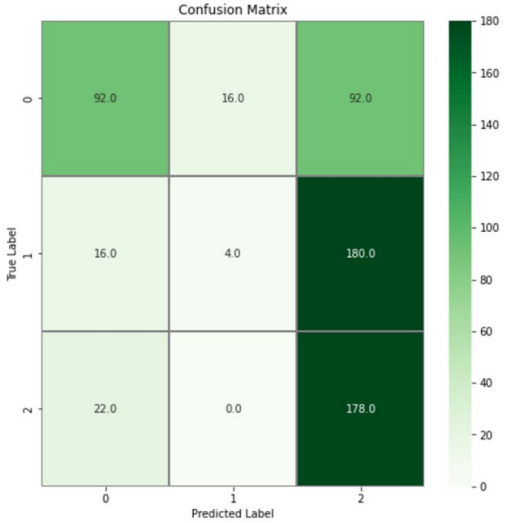
Model	Learning rate & Optimizer	Batch Size	Loss	Accuracy (%)	F-1 score
Shallow CNN	0.0001 & RMSProp	16	1.2829	45.67	0.38
ResNet- 50	0.00001 & RMSProp	8	0.9248	70.67	0.71
EfficientNetB4	0.00001 & Adam	16	0.7110	65.17	0.66
VGG16	0.0001 & RMSProp	4	0.8507	74	0.73

Results of testing our trained models on Potsdam dataset

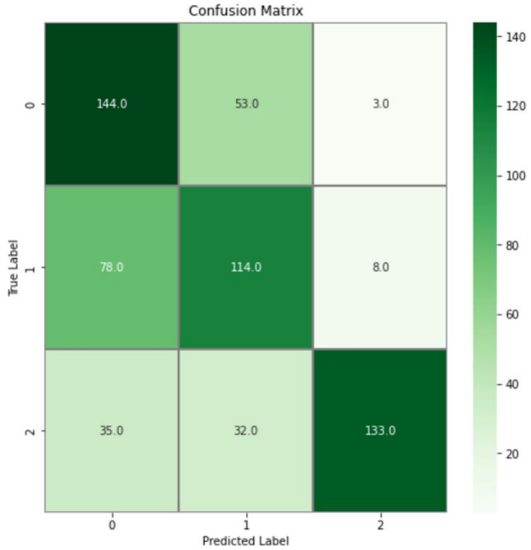
ResNet50



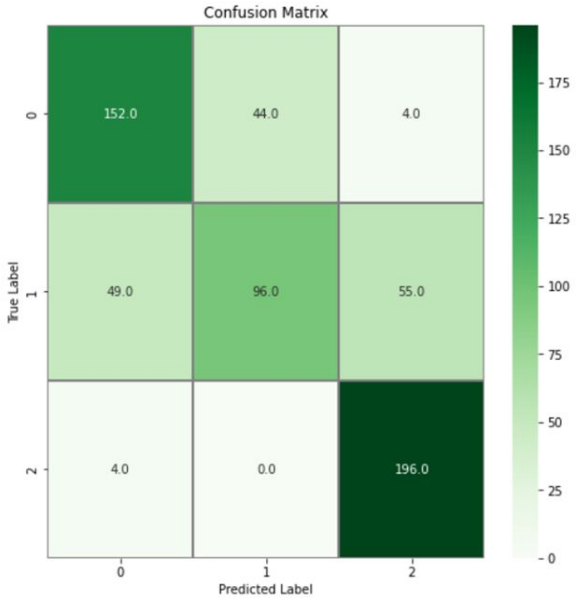
Shallow CNN



EfficientNetB4

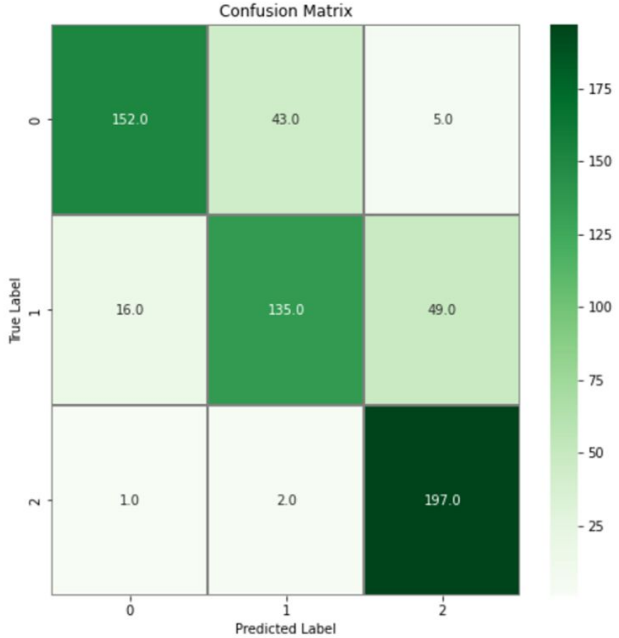


VGG16



Majority Voting

```
Shape of combined results: (600, 3)
```




```
0.8066666666666666
```

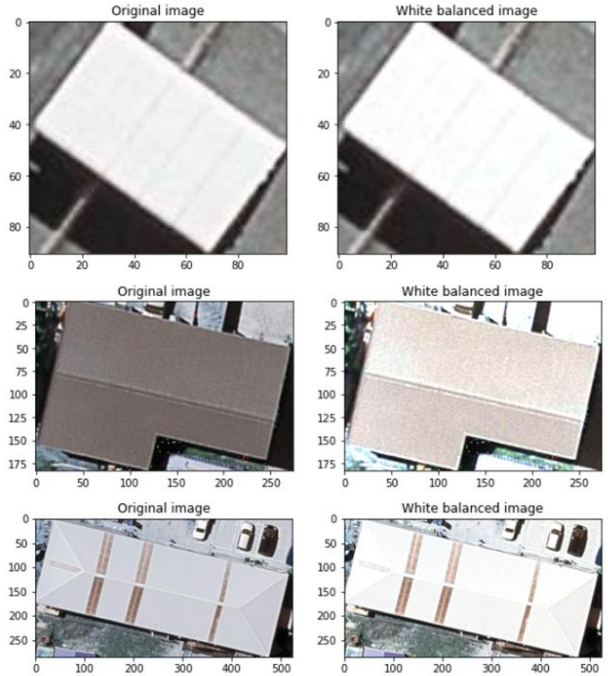
2.4 Boundary Detection

2.4.1 White Balancing

- White balance is used to remove any coloured haze from an image so that it seems to be under white light.
- 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
```



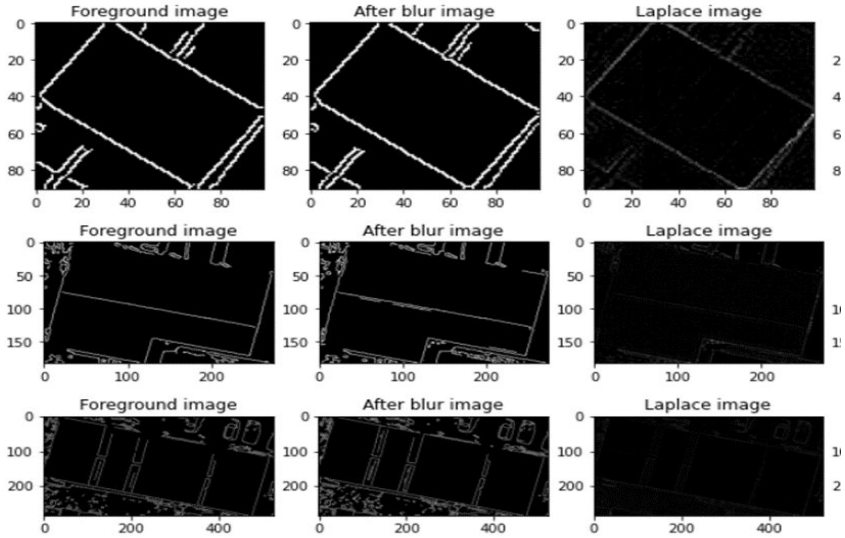
Images after performing white balancing to remove haze

2.4.2 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 33 is used with = 1.5

2.4.3 Auto Canny Edge Detection

- Multi-staged algorithm that includes noise reduction, finding intensity gradients and hysteresis suppression
- 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.



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:

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

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

METRICS FOR EVALUATION

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.

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

METRICS FOR EVALUATION

For roof type classification:

- **Classification Report**

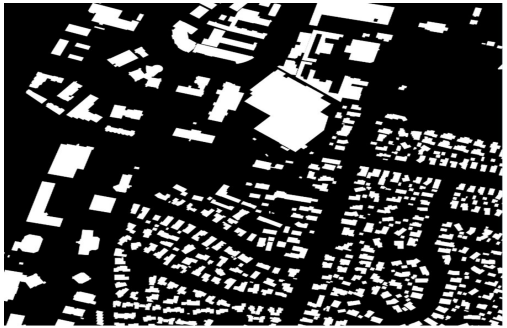
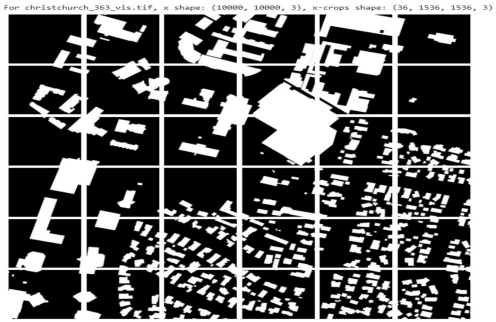
The classification report is used to measure the quality of predictions from a classification algorithm. Precision, Recall and F1 scores are calculated on a per-class basis based on True Positives, True Negatives, False Positives, False Negatives. Here, we calculate the above values for each of the classes,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

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \qquad Recall = \frac{TruePositive}{TruePositive + FalseNegative} \qquad F1 = 2.\frac{Precision \times Recall}{Precision + Recall}$$

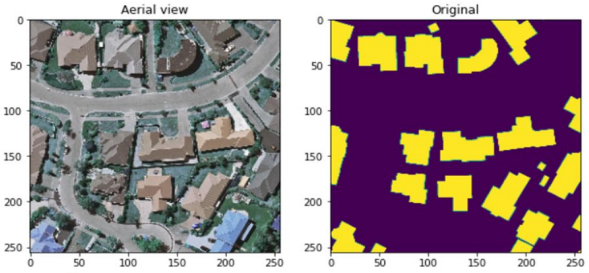
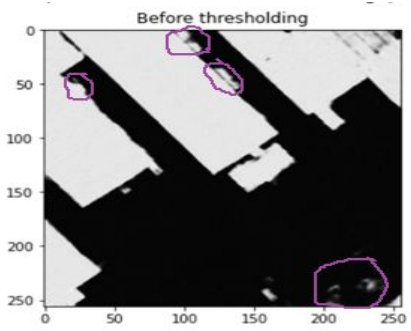
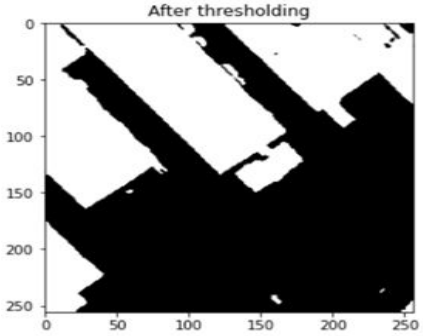
- **AUC-ROC**

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 classification algorithm.

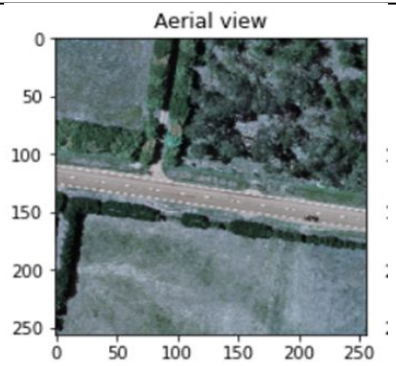
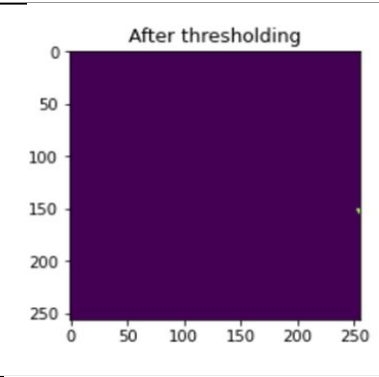
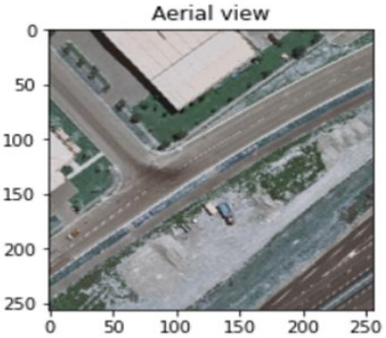
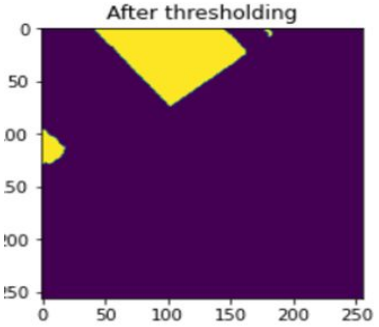
TEST CASES

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_01	Clipping of aerial satellite images into (1536 * 1536) dimensions.		
TC_02	Clipping of ground truth images into (1536 * 1536) dimensions.		

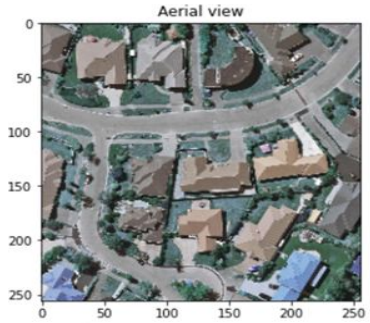
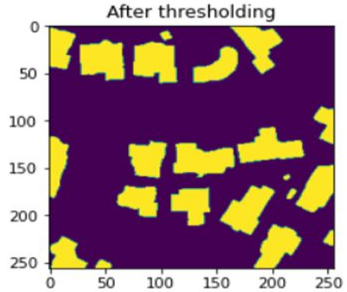
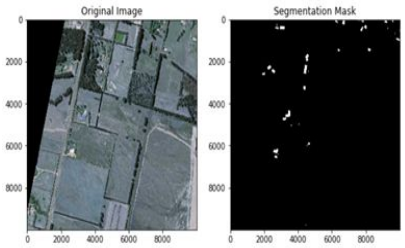
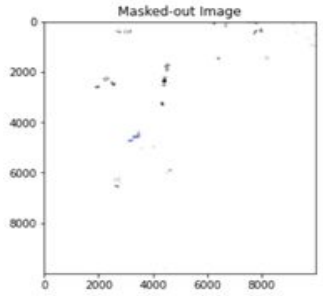
TEST CASES

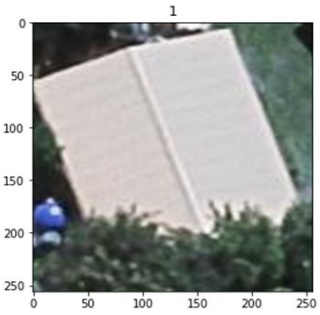
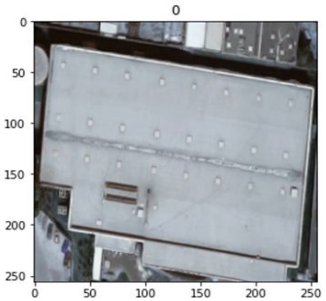
TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_03	Normalisation		
TC_04	Applying Threshold		

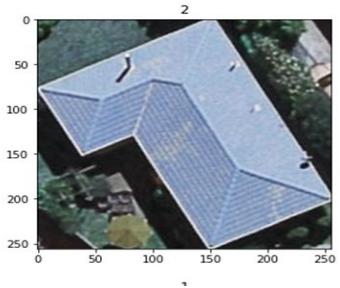
TEST CASES

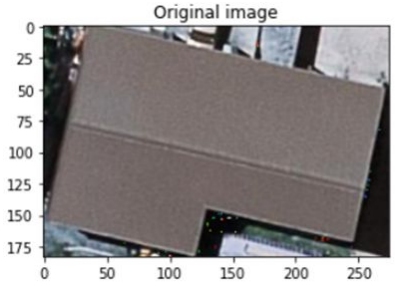
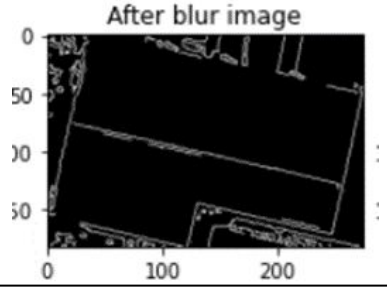
TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_05	Building segmentation results- with no buildings.		
TC_06	Building segmentation of a single building.		

TEST CASES

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
TC_07	Building segmentation of various buildings in a single image.		
TC_08	Rooftop extraction by background subtraction		

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
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

TEST CASE ID	TEST CASE DESCRIPTION	TEST INPUT	TEST OUTPUT
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.

REFERENCES

[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.

THANK YOU

	A	B	C	D	E	F	G	H	I	J	K	L
1	**Model Name**	**Architecture**	**Numpy or Image Generator**	**Epochs**	**Batch size**	**Learning Rate**	**No of images**	**Performance Metrics**	**Colab or Kaggle**	**Filename**	**Comments**	**Results**
2	u-net-1000images-baseline	U-Net (All U-Net images are of dimensions 256 * 256)	Numpy	10	8	0.01	1008	IOU, Accuracy	Colab	Basic U-Net	Scaling done by directly by 255	IOU - 18.06, Accuracy - 95.81
3	unet-youtube-1000images-10 epochs	U-Net	Numpy	10	16	0.01	1008	IOU, Dice Coefficient, Accuracy	Colab	Youtube U-Net	Used MinMaxScaler, Referred Digital Sreeni video	IOU - 64.73, Dice Coefficient - 78.43 Accuracy - 94.84
4	unetmcc-youtube-1000image s-10epochs	U-Net	Numpy	10	16	0.01	1008	IOU, Dice Coefficient, MCC, Accuracy	Colab	Youtube U-Net	Used MinMaxScaler, Added MCC also as performance metrics, Referred Digital Sreeni video	IOU - 68.22, Dice Coefficient - 80.99, MCC - 14.40, Accuracy - 95.37
5	resumable-model	U-Net	Numpy	15	8	0.001	1008	IOU, Dice Coefficient, MCC, Accuracy	Colab	Youtube U-Net	Same as above, Tried How to start, stop resume training of a model using keras-buoy	IOU - 70.36, Dice Coefficient - 82.34, MCC - NaN, Accuracy - 94.98
6	unet2-1000images-100epoch s	U-Net	Numpy	100	8	0.0001	1008	IOU, Dice Coefficient, MCC, Accuracy	Colab	U-Net Final	Tried resumable models too but that wasn't done properly.	IOU - 88.40, Dice Coefficient - 93.79, MCC - 51.68, Accuracy - 98.22, Loss - 0.0356
7	unet2-1000images-130epoch s	U-Net	Numpy	30 (Continue training from the previous model) 100+30=130	8	0.00001	1008	IOU, Dice Coefficient, MCC, Accuracy	Colab	U-Net Final	Continued training from the previous model by loading the weights and decreasing the learning rate. Trained for 30 more epochs.	IOU - 90.57, Dice Coefficient - 95.02, MCC - 56.88, Accuracy - 98.48, Loss - 0.0279
8	1500images-npy-100epochs	U-Net	Numpy	100	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Kaggle	1500 Numpy MultiRes Version 1 - Numpy MultiRes,U-Net	UNet with numpy array. Ran faster than in Colab. Took 48s for each epoch in Kaggle compared to 2mins in Colab.	IOU - 86.81, Dice Coefficient - 92.83, MCC - 48.97, Accuracy - 98.28, Loss - 0.0339
9												
10	focalloss-30epochs	MultiRes	Numpy	100	8	0.0001	1008	IOU, Dice Coefficient, MCC, Accuracy	Kaggle	Keras Data Generator Version 1 - multires-30	MultiRes with binary focal loss instead of binary cross entropy loss.	IOU - 32.33, Dice Coefficient - 48.43, MCC - 0.88, Accuracy - 97.73, Loss - 0.0422
11	1500images-30epochs	MultiRes	Numpy	30 (256*192)	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Kaggle	1500 Numpy MultiRes Version 1 - Numpy MultiRes,U-Net	Tried MultiRes UNet with numpy array.	IOU - 30.21, Dice Coefficient - 45.87, MCC - 0.89, Accuracy - 97.53, Loss - 0.3288
12	multires-100epochs	MultiRes	Numpy	100 (256*192	8	0.01	500	IOU, Dice Coefficient, MCC, Accuracy	Colab	MultiRes UNet v1	MultiRes UNet (original filter) not as in base paper. The output segmentation masks results seem somewhat good and performance has improved slightly.	IOU - 39.42, Dice Coefficient - 56.53, MCC - 2.77, Accuracy - 94.44, Loss - 0.2037
13	Dropout-500images-30epoch s-MultiRes	MultiRes	Numpy	30 (256*192)	8	0.01	500	IOU, Dice Coefficient, MCC, Accuracy	Kaggle	1500 Numpy MultiRes Version 3 - Dropout-MultiRe s-500	Used Dropout(0.3) in conv_2d_bn function. 56 dropouts	IOU - 30.58, Dice Coefficient - 46.39, MCC - 2.07, Accuracy - 90.41, Loss - 0.3374

14	multires-pro-500images-bp-50epochs	MultiRes	Numpy	50 (256*256)	8	0.001	500	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Pro - MultiRes UNet	Tried in Colab Pro with filters as mentioned in base paper & image dimensions (256*256)	IOU - 39.17, Dice Coefficient - 56.27, MCC - 2.79, Accuracy - 95.21, Loss - 0.3917
15	multires-pro-allimages-40epochs	MultiRes	Numpy	40 (256*256)	4	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Pro - MultiRes UNet	Original MultiRes with filters (32, 64, 128, 256). Excellent results for validation set compared to best of U-Net but IOU is still less. (Revelation found). Test dataset contains image with no buildings but training dataset has no such image. Thus, need to include no building images in training set as well.	IOU - 36.33, Dice Coefficient - 53.06, MCC - 2.39, Accuracy - 93.83, Loss - 0.1782
16	multires-pro-1500images-bp-50epochs	MultiRes	Numpy	50	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Revelation - Pro - MultiRes UNet	https://github.com/nibtehaz/MultiResUNet/issues/16 In this model, IOU formula is as previously used. But tried the new IOU formula given in the above link (rounding off) and IOU worked well for val and test images. BP Filters are used. Same for Dice coefficient.	IOU - 34.15, Dice Coefficient - 50.56, MCC - 1.21, Accuracy - 98.46, Loss - 0.2535
17								IOU, Dice Coefficient, MCC,		Revelation - Pro		IOU - 38.84, Dice Coefficient - 55.86, MCC - 2.31, Accuracy - 98.54,

	A	B	C	D	E	F	G	H	I	J	K	L
16	multires-pro-1500images-bp-50epochs	MultiRes	Numpy	50	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Revelation - Pro - MultiRes UNet	https://github.com/nibtehaz/MultiResUNet/issues/16 In this model, IOU formula is as previously used. But tried the new IOU formula given in the above link (rounding off) and IOU worked well for val and test images. BP Filters are used. Same for Dice coefficient.	IOU - 34.15, Dice Coefficient - 50.56, MCC - 1.21, Accuracy - 98.46, Loss - 0.2535
17	multires-pro-allimages-100epo	MultiRes	Numpy	100	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Revelation - Pro - MultiRes UNet	Excellent results.	IOU - 38.84, Dice Coefficient - 55.86, MCC - 2.31, Accuracy - 98.54, Loss - 0.1713
18	multires-pro-1500images-bp-1(	MultiRes	Numpy	100	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Final v1 MultiRes UNet	Base paper filters - final version 1 with changed formula.	IOU - 93.53, Dice Coefficient - 96.25, MCC - 95.87, Accuracy - 98.51, Loss - 0.1734
19	multires-pro-allimages-100epochs	MultiRes	Numpy	100	8	0.0001	1548	IOU, Dice Coefficient, MCC, Accuracy	Colab Pro	Final v1 MultiRes UNet	Modification to base paper by changing the filters	IOU - 93.76, Dice Coefficient - 96.66, MCC - 96.16, Accuracy - 98.63, Loss - 0.1743
20												
21												
22												
23												

POTSDAM SAMPLE IMAGES

Flat1

Flat2

Flat3

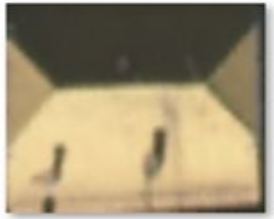
Flat4

Gable28

Gable29

Gable30

Gable31



Hip54

Hip55

Hip56

Hip57

Untitled spreadsheet

File Edit View Insert Format Data Tools Extensions Help Last edit was on April 10

100% $ % .0 .00 123 Default (Ari... 10

A1 S.No.

	A	B	C	D
1	S.No.	Image Name	Type	Annotated or not
2	1	1.tif	Gable	Yes
3	2	3.tif	Complex	
4	3	6.tif	Flat	
5	4	8.tif	Gable	Yes
6	5	15.tif	Gable	To be deleted
7	6	16.tif	Gable	Yes
8	7	18.tif	Flat	
9	8	27.tif	Flat	
10	9	29.tif	Gable	Yes
11	10	30.tif	Gable	Yes
12	11	35.tif	Gable	Yes
13	12	christchurch_36_3.tif	Hip	
14	13	christchurch_36_5.tif	Gable	Yes
15	14	christchurch_36_6.tif	Gable	Yes
16	15	christchurch_36_16.tif	Complex	
17	16	christchurch_36_18.tif	Complex	
18	17	christchurch_36_29.tif	Flat	
19	18	christchurch_36_30.tif	Gable	Yes
20	19	christchurch_36_31.tif	Gable	Yes
21	20	christchurch_36_33.tif	Gable	Yes
22	21	christchurch_36_37.tif	Complex	

Rooftop_Types

Untitled spreadsheet

File Edit View Insert Format Data Tools Extensions Help Last edit was on April 10

100% $ % .0 .00 123 Default (Ari... 10

A1 S.No.

	A	B	C	D
1348	1347	yeyyO-31924	Flat	
1349	1348	yfryl-86650	Flat	
1350	1349	YFWSK-53068	Flat	
1351	1350	YkzLz-39951	Flat	
1352	1351	ylBSJ-55543	Flat	
1353	1352	YlJcC-59362	Flat	
1354	1353	YmXQu-43633	Flat	
1355	1354	ynpbI-89067	Flat	
1356	1355	YoQof-93452	Flat	
1357	1356	ypBoe-13811	Flat	
1358	1357	ySYoW-48498	Flat	
1359	1358	yvcqD-32337	Flat	
1360	1359	yVFhd-99563	Flat	
1361	1360	yWZIF-99489	Flat	
1362	1361	YyKlj-24618	Flat	
1363	1362	yZZit-35376	Flat	
1364	1363	zawos-61611	Flat	
1365	1364	ZNrwf-45215	Flat	
1366	1365	znsrP-97157	Flat	
1367	1366	ZorrT-16821	Flat	
1368	1367	ZQQrC-78932	Flat	
1369	1368	ztbuf-94418	Flat	

Rooftop_Types

A1 | Model Name

	A	B	C	D	E	F	G	H	I	J	K	
1	**Model Name**	**Dataset**	**Architecture**	**Epochs**	**Learning Rate**	**Optimizer**	**Batch Size**	**Performance Metrics**	**Filename**	**Comments**	**Results**	**Resu**
2	./ResNet50-finetuning-bs-8	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 319	ResNet50	100	0.00001	RMSProp	8	Accuracy, Classification Report	Classification with mod data - Version 0 - resnet & efficientnet	Gable misclassified as flat in potsdam dataset	Loss: 0.0871 Accuracy: 0.9753 Val Loss: 0.3376 Val Accuracy: 0.9815 Test Loss: 0.7633 Test Accuracy: 0.8667 F1-score: 0.86 Recall: 0.86 Precision: 0.86	Loss: Accu 300 ir class
3	./EfficientNetB4-finetuning-bs16-RMS	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 319	EfficientNetB4	100	0.00001	RMSProp	16	Accuracy, Classification Report	Classification with mod data - Version 0 - resnet & efficientnet	Gable misclassified as flat in potsdam dataset	Loss: 0.3233 Accuracy: 0.8756 Val Loss: 0.2548 Val Accuracy: 0.92593 Test Loss: 0.3158 Test Accuracy: 0.8889 F1-score: 0.89 Recall: 0.89 Precision: 0.89	Loss: Accu 300 ir class
4	./EfficientNetB4-finetuning-	Modified dataset: rooftype-dataset-mod						Accuracy,	Classification with mod data - Version 1 - EfficientNet Adam-		Loss: 0.1280 Accuracy: 0.9552 Val Loss: 0.1322 Val Accuracy: 0.9815 Test Loss: 0.3158 Test Accuracy: 0.8889 F1-score: 0.91 Recall: 0.91	Loss: Accu 300 ir

A1 | Model Name

	A	B	C	D	E	F	G	H	I	J	K	
5	./VGG16-finetuning-RMS-bs4	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 320	VGG16	100	0.00001	RMSProp	4	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 2 - VGG16 with ROC	Added Au-ROC metrics. Results are compariively not so good for PotsDam testing	Loss: 0.1011 Accuracy: 0.9705 Val Loss: 1.0823 Val Accuracy: 0.9259 Test Loss: 1.5621 Test Accuracy: 0.8444 F1-score: 0.84 Recall: 0.85 Precision: 0.84 ROC_AUC score for 3 models: {0: 0.9519230769230769, 1: 0.9453781512605042, 2: 0.9311111111111111}	Loss: Accu 300 ir class
6	./VGG16-finetuning-RMS-bs4	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 321	VGG16	100	0.00001	RMSProp	4	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 3 - TTA for VGG16	TTA has improved accuracy from 57.33 to 59	Loss: 0.1011 Accuracy: 0.9705 Val Loss: 1.0823 Val Accuracy: 0.9259 Test Loss: 1.5621 Test Accuracy: 0.8444 F1-score: 0.84 Recall: 0.85 Precision: 0.84 ROC_AUC score for 3 models: {0: 0.9519230769230769, 1: 0.9453781512605042, 2: 0.9311111111111111}	Loss: Accu 300 ir class accur

A1 | Model Name

	A	B	C	D	E	F	G	H	I	J	K	
8	./CNN-RMSProp-50epochs	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 321	Shallow CNN	50	0.0001	RMSProp	16	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 5 - all models 50 epochs	Nearly all gable images are classified as hip.	Loss: 0.7179 Accuracy: 0.6736 Val Loss: 0.6047 Val Accuracy: 0.8148 Test Loss: 0.8446 Test Accuracy: 0.7111 F1-score: 0.71 Recall: 0.73 Precision: 0.72 ROC_AUC score for 3 models: {0: 0.8716577540106951, 1: 0.7470355731225298, 2: 0.8560606060606061}	Loss: Accur 300 ir class accur
9	./ResNet50-50epochs-bs8	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 322	ResNet50	50	0.00001	RMSProp	8	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 5 - all models 50 epochs	Some gable are misclassified as flat. Better results than 100 epochs.	Loss: 0.1688 Accuracy: 0.9427 Val Loss: 0.2977 Val Accuracy: 0.9444 Test Loss: 0.8303 Test Accuracy: 0.8444 F1-score: 0.84 Recall: 0.85 Precision: 0.84 ROC_AUC score for 3 models: {0: 0.868983957219251 4, 1: 0.9169960474308 3, 2: 1.0}	Loss: Accur 300 ir class accur

A1 | Model Name

	A	B	C	D	E	F	G	H	I	J	K	
10	./EfficientNetB4-50epochs-bs16-RMS	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 323	EfficientNetB4	50	0.00001	RMSProp	8	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 5 - all models 50 epochs	Some gable are misclassified as flat. Better results than 100 epochs.	Loss: 0.4292 Accuracy: 0.8189 Val Loss: 0.2895 Val Accuracy: 0.9074 Test Loss: 0.7727 Test Accuracy: 0.6889 F1-score: 0.70 Recall: 0.72 Precision: 0.79 ROC_AUC score for 3 models: {0: 0.8235294117647058, 1: 0.8339920948616 6, 2: 0.9848484848484848}	Loss: Accur 300 ir class accur
11	./EfficientNetB4-50epochs-bs16-Adam	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 324	EfficientNetB4	50	0.00001	Adam	8	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 5 - all models 50 epochs	Flat very accurately classified. Gable miscalsified as flat. Better results than 50 epochs.	Loss: 0.3815 Accuracy: 0.8567 Val Loss: 0.3477 Val Accuracy: 0.8889 Test Loss: 0.5437 Test Accuracy: 0.8222 F1-score: 0.82 Recall: 0.82 Precision: 0.83 ROC_AUC score for 3 models: {0: 0.8957219251336899, 1: 0.9011857707509882, 2: 0.9873737373737375}	Loss: Accur 300 ir class accur

	A	B	C	D	E	F	G	H	I	J	K	
12	./VGG16-50epochs-RMS-bs	Modified dataset: rooftype-dataset-mod Flat - 374 Gable - 429 Hip - 325	VGG16	50	0.00001	RMSProp	4	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 5 - all models 50 epochs	Flat very accurately classified. Gable miscalsified as flat. Better results than 50 epochs.	Loss: 0.1585 Accuracy: 0.9440 Val Loss: 0.2503 Val Accuracy: 0.9630 Test Loss: 0.8224 Test Accuracy: 0.8000 F1-score: 0.78 Recall: 0.78 Precision: 0.80 ROC_AUC score for 3 models: {0: 0.8556149732620322, 1: 0.9031620553359684, 2: 1.0}	Loss: Accu 300 ir class accur
13												
14	/CNN-RMSProp-50epochs							Accuracy, Classification Report	Classification with mod data - Version 6 - mod-dataset-2	Gable misclassified as Hip. But gable and flat	Loss: 0.7212 Accuracy: 0.6620 Val Loss: 0.6658 Val Accuracy: 0.6852 Test Loss: 0.8978 Test Accuracy: 0.5778 F1-score: 0.58 Recall: 0.68 Precision: 0.58 ROC_AUC score for 3 models: {0: 0.779797979797798, 1: 0.6957707509881423, 2:	Loss: Accu 300 ir class

	A	B	C	D	E	F	G	H	I	J	K	
15	./ResNet50-50epochs-bs8-mod2	Mod dataset 2	ResNet50	50	0.00001	RMSProp	8	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 6 - mod-dataset-2 50epochs	Very good classification in PotsDam.	Loss: 0.1331 Accuracy: 0.9488 Val Loss: 0.4049 Val Accuracy: 0.9444 Test Loss: 0.5135 Test Accuracy: 0.8889 F1-score: 0.89 Recall: 0.90 Precision: 0.89 ROC_AUC score for 3 models: {0: 0.9696969696969697, 1: 0.9624505928853756, 2: 0.9946524064171123}	Loss: Accu 300 ir class accur
16	./EfficientNetB4-50epochs-bs16-RMS-mod2	Mod dataset 2	ResNet51	50	0.00001	RMSProp	16	Accuracy, Classification Report, AU-ROC	Classification with mod data - Version 6 - mod-dataset-2 50epochs		Loss: 0.1331 Accuracy: 0.9488 Val Loss: 0.4049 Val Accuracy: 0.9444 Test Loss: 0.5135 Test Accuracy: 0.8889 F1-score: 0.89 Recall: 0.90 Precision: 0.89 ROC_AUC score for 3 models: {0: 0.9696969696969697, 1: 0.9624505928853756, 2: 0.9946524064171123}	Loss: Accu 300 ir class accur