MSiA400

Team: Grand Teton

Team Members:

Bhavya Kaushik

Daniel Wang

Jing Ren

Zach Zhu

Build Change House Retrofit **Classification**

2019 Fall MSiA 400 Project

1. **Introduction**

The team is working with Build Change to train and implement artificial intelligence to perform structural analysis of houses in order to classify whether they are suitable for retrofitting after natural disasters. Responders at Build Change face the challenge of rapidly assessing affected areas and helping communities recover after natural disasters. Retrofitting is a cost-effective and rapid solution for preserving a homeowners' damaged home; however, sending engineers on-site for structural evaluation is costly and inefficient. As a result, Build Change is considering building an app that will use a Machine Learning model, coupled with Computer Vision techniques, and allow users to assess whether their house is a good candidate for retrofitting.

The team was given a large training sample (10,000 images) of artificial images of houses based on CAD modelling of the houses' structure and built a classification model based on these training images using logistic regression and XGBoost. In order to apply the model on real-world images, the team primarily used packages OpenCV in python to extract feature vectors (# openings, # levels, house sizes, etc.) from the real-world images, and feed these feature vectors into the classification model to get final classification results. We ended up predicting 160 images of houses to "go" and 32 images of houses to "NoGo".

2. **Tasks**

- Create a feature vector based on syntactic images by using modules including OpenCV
- Create a classification model for syntactic images
- Evaluate the model on real world images
- Improve feature engineering to improve the performance on real world images

3. **Feature Vector Extraction**

Image Pre-processing

We are interested in the features of the house that gives a comprehensive indication of its structural damage. Before we apply any feature detection methods on the images, we preprocessed the images in the following ways. First, we turned the images into grayscale to avoid dealing with different colors within the images. For the syntactic images, we threshold the images using binary thresholding since the images have colors and line segments that are clearly separated.

For the real images, we took a different and more complex approach in terms of pre-processing. We used Gaussian adaptive thresholding instead of binary thresholding since the different areas of the real images are affected by varying lighting conditions, and a single global threshold value is not optimal. Gaussian threshold value is the weighted sum of neighbourhood values where weights are a gaussian window, and is more suitable for the case of the real images.

For the real images we also used bilateral filtering for smoothing and reducing noise. Bilateral filtering is an effective tool to blur the images while still keeping the edges sharp. This approach helps canny edge-detection (performed in later steps) in OpenCV, and gives us better line and contour detection.

Level Detection

The first feature of the buildings is the number of levels, since we believe it is significantly correlated to the house's repairability. It is also noted in the Excel sheet "Retrofit_Module_Inputs_Table_Questionnaire" that 15 - 20% of the houses that went through retrofitting have 1 story, while 60 - 70% have 3 stories. We extracted this feature in the syntactic images by first using OpenCV's canny edge detection to detect all the significant edges, and then using Hough Line Transform to detect the horizontal straight lines that exceeds 80% of the total width of the images. We also made sure to delete repeated lines and line detected from the roof (we only need floor lines to count the number of floors).

For the real images we took a different approach to detect horizontal lines in the images. Since most of the images are not photographed on a frontal angle, it is not best practice to find the horizontal lines. Also unlike generated images, there often isn't a clear line separating two floors in the real images. So we can only get the number of floors by counting the number of vertically aligned windows. Therefore we detected lines based on contours. Contours are curves joining all the continuous points having the same color or intensity. We approximated polygons from contours and filtered out those contours from each image which represented the edges of doors and windows or demarcations separating different floors. We then extracted lines from those polygons. We omitted lines with slope greater than 15 degrees, and lines which are too close to the image borders and lines which are too small in length to further reduce the noise. Finally we counted only those lines which weren't so close to each other that they end up being on the same floor. Thus we got a decent estimate of the number of floors in each building. A few visualizations of lines detected on real and generated images, are given in the Appendix.

Opening Detection

We then worked on detecting the number of openings of a house. Note that openings include both manufactured openings like doors, windows, and openings caused by natural disasters such as holes on walls. We extracted the contours in the image using the OpenCV method findContours, and then using approxPolyDP we are able to approximate the contours into convex polygons, we then filter the obtained polygons based on the number of their sides. Since we are interested in mostly quadrilateral openings (doors, windows, etc.) we only considered contours with number of sides between 3 and 6. Further, we found out the area of the contours and filtered out those contours which had very small area (less than 0.1% of the image area) or very large area (more than 15% of the image area) with respect to the image , we also filtered polygons which were too close to the border of the image. The resulting contours overlapped very well with the windows and doors and hence gave a good estimate of the number of openings in a building. A few visualizations of the contours detected on real and generated images, are given in the Appendix.

Other Derived Attributes

We then derived several additional metrics to further differentiate the images of houses from each other. We noted that although some houses have a lot of openings detected, they have a lot of levels as well, which makes the number of openings on each level relatively low. We therefore

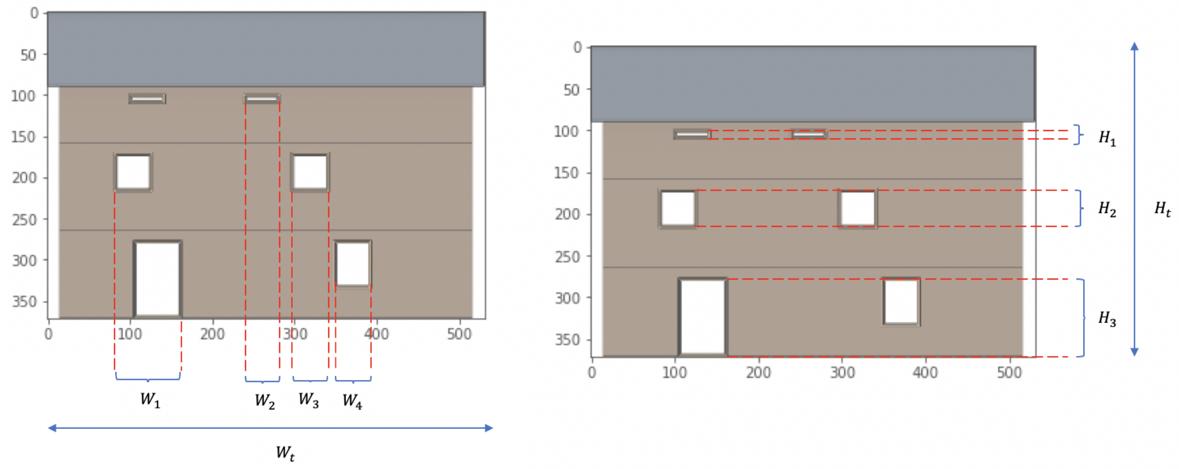


Figure 1 *Fractional width and height illustration*

created the feature **"fraction_width"** to represent the fractional width of the openings compared to the width of the house. In the example below, the calculation is (W1 + W2 + W3 + W4) / Wt. We repeat this process on the y- axis to get a similar feature **"fraction_height"**.

We also accounted for the impact that number of floors have on the total width of openings of the building. Therefore, we decided to add two additional predictors, **"avg_fraction_width"** and **"aggregate_fraction_height"**, where we take the sum of all windows widths (heights) (divided by the number of floors) and find the ratio of that sum to the overall width (height) of the building.

We further calculated the fraction of the total area of the openings of the house to the entire area of the image. We approximate the area of the building using the sizes of the image, since extracting the contour of the house itself is not easily done with all the foreign objects covering up the buildings.

4. **Model Fitting**

To test whether our feature vector (nine features) has enough predictive power classifying houses into Go and NoGo, we first fit simple logistic regression model on the generated images. We randomly split the data into 1000 test images (12%) and 7371 training images (88%). Since some of the predictors that we are using are in different units, we normalized the feature vectors. We used 5-fold validation to obtain the best penalty and inverse regularization parameter C. The best model yielded 78.81% accuracy (Table 1).

Logistic			**XGBoost**		
Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
71.40%	78.00%	74.55%	82.43%	79.75%	81.07%

Table 1 *Evaluation of the two models*

We then ran an XGBoost model on the training images and yielded an accuracy of 85.71%, a significant improvement from our logistic regression model. The model also yielded an F1-score of 81% which is higher than the logistic regression model. We decided to look at the feature importance plot from XGBoost model to see which feature is most useful in constructing the boosted decision trees.

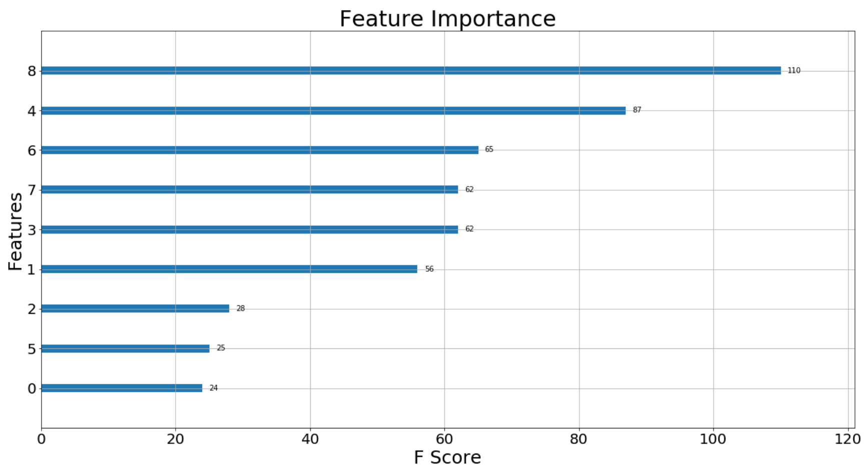


Figure 2 *Feature Importance (See Appendix 1 for feature rationale)*

We've noticed that though contrary to our prior belief, the number of levels of the houses (predictor "0") is the least important compared to other predictors in the model. However, after we dropped this predictor the auc score dropped from 0.8752 to 0.8644. Therefore we decided to keep the predictor in the model.

5. **Real Images Classification**

We used the XGBoost model selected above to classify the real images. Since we don't have labels of the real images (whether they are chosen to be retrofitted), we decided to manually label the images based on the rules set out in the excel sheet **"Retrofit_Module_Inputs_Table_Questionnaire"**. Namely:

- sum of all windows' widths (on the same floor) must be no more than 35% of overall length of the building
- minimum space of 1000 mm between windows, 600 mm between a door and a window, 800 between an opening and wall edge

- Maximum overall length of 9000 mm
- Maximum floor-to-floor height of 2100 mm
- Maximum wall height of 1200 mm

Comparing the manually set labels and the predicted labels given by XGBoost model, we recorded a precision of 0.5, recall of 0.15, and F1-score of 0.24. In the case of Build Change, we are more concerned with precision over recall, since the cost of False Negatives is of the primal concern (falsely classifying a building to not suitable for retrofitting). Therefore, we are more concerned with the value of precision rather than recall. However, we need more consultation from Build Change in order to quantify the cost of False Positives and False Negatives in order to come up with a model that gives a more optimal classification.

6. **Conclusions**

For this project, we trained a logistic regression model and a XGBoost model on the training set of images after extracting the feature vectors from the syntactic images, and decided to use the XGBoost model after comparing their evaluation metrics. We classified 160 images of houses to “Go” and 32 images to “NoGo”.

The quality of some real images may not good enough, such as no front, trees hide the majority of the building from view, and no reference to measure the size of the houses. Feature extraction worked fine for most of the metrics we selected including number of openings, number of levels, area of openings, etc., and we were able to get a similar distribution of the feature vectors of the real images compared to the syntactic images. However, we weren’t able to filter out the contour of the outside periphery of the house itself, since some of the buildings are covered by other objects such as plants and animals, and therefore could not provide an accurate measure of the area of the front facet of the house. With more granular search and identification of the features of the houses we might be able to capture more information and provide a more predictive model.

In the future, we will also try Neural Style Transfer to transform the styles of the training images (colors, lines, etc.) onto the real images, so that the feature extraction methods we implement on the training images can be more conveniently reused on the real images.

7. **Reference**

1. *Image Segmentation using OpenCV - Extracting specific Areas of an image*, 2019, https://circuitdigest.com/tutorial/image-segmentation-using-opencv, Accessed 10 Dec. 2019
2. *OpenCV Modules*, https://docs.opencv.org/3.4/, Accessed 5 Dec. 2019

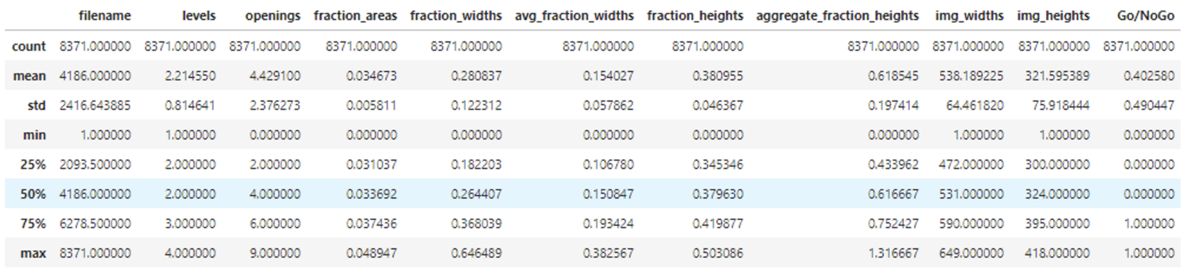
8. **Appendix**

Appendix 1: Features

Feature Name	**Rationale**
level	Number of levels/floors
opening	Number of openings
fraction_area	Sum of all openings' areas / overall area of image
fraction_widths	Sum of all openings' heights (without overlap) / width of the house
avg_fraction_widths	Sum of all openings' heights / number of floors / width of the house
fraction_heights	Sum of all openings' heights (without overlap) / height of the house
aggregate_fraction_heights	Sum of all openings' heights / number of floors / height of the house
img_widths	House width
img_heights	House height

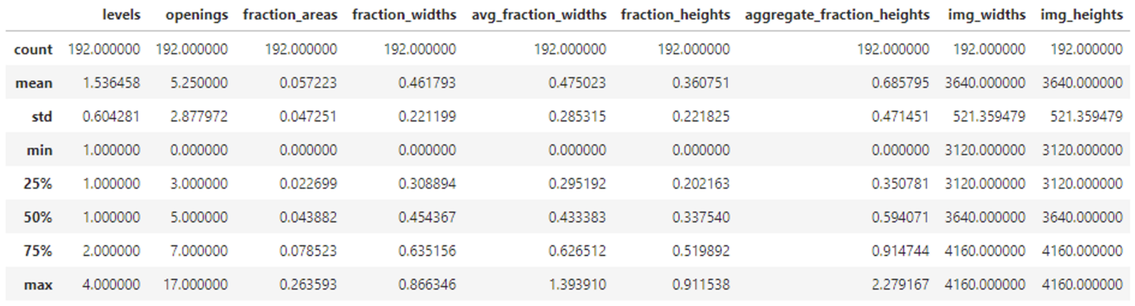
Appendix 2: Summary statistics of generated features

1. Generated data



	filename	levels	openings	fraction_areas	fraction_widths	avg_fraction_widths	fraction_heights	aggregate_fraction_heights	img_widths	img_heights	Go/NoGo
count	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000	8371.000000
mean	4186.000000	2.214550	4.429100	0.034673	0.280837	0.154027	0.380955	0.618545	538.189225	321.595389	0.402580
std	2416.643885	0.814641	2.376273	0.005811	0.122312	0.057862	0.046367	0.197414	64.461820	75.918444	0.490447
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000
25%	2093.500000	2.000000	2.000000	0.031037	0.182203	0.106780	0.345346	0.433962	472.000000	300.000000	0.000000
50%	4186.000000	2.000000	4.000000	0.033692	0.264407	0.150847	0.379630	0.616667	531.000000	324.000000	0.000000
75%	6278.500000	3.000000	6.000000	0.037436	0.368039	0.193424	0.419877	0.752427	590.000000	395.000000	1.000000
max	8371.000000	4.000000	9.000000	0.048947	0.646489	0.382567	0.503086	1.316667	649.000000	418.000000	1.000000

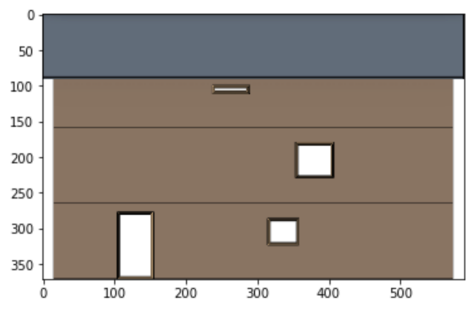
2. Real data



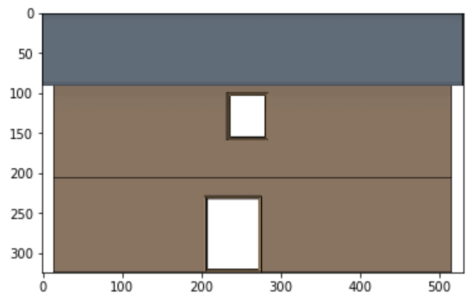
	levels	openings	fraction_areas	fraction_widths	avg_fraction_widths	fraction_heights	aggregate_fraction_heights	img_widths	img_heights
count	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000	192.000000
mean	1.536458	5.250000	0.057223	0.461793	0.475023	0.360751	0.685795	3640.000000	3640.000000
std	0.604281	2.877972	0.047251	0.221199	0.285315	0.221825	0.471451	521.359479	521.359479
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3120.000000	3120.000000
25%	1.000000	3.000000	0.022699	0.308894	0.295192	0.202163	0.350781	3120.000000	3120.000000
50%	1.000000	5.000000	0.043882	0.454367	0.433383	0.337540	0.594071	3640.000000	3640.000000
75%	2.000000	7.000000	0.078523	0.635156	0.626512	0.519892	0.914744	4160.000000	4160.000000
max	4.000000	17.000000	0.263593	0.866346	1.393910	0.911538	2.279167	4160.000000	4160.000000

Appendix 3: Feature Vector Examples

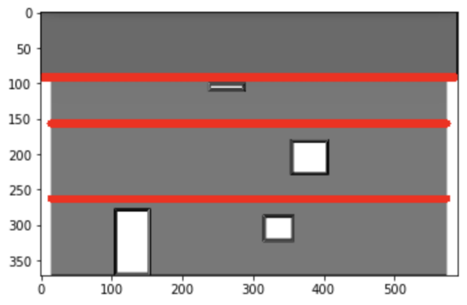
A. Level Detection (Syntactic Images)

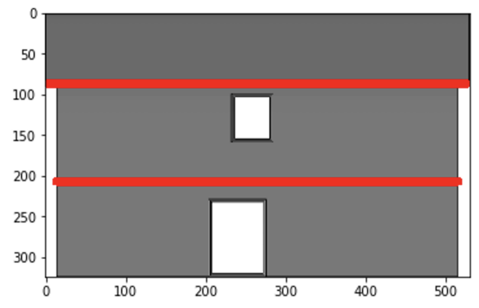


```
number of levels =  3
```



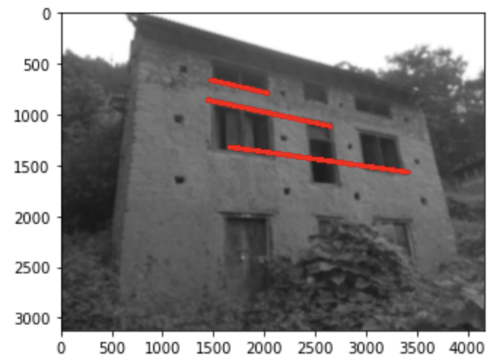
```
number of openings =  2
```



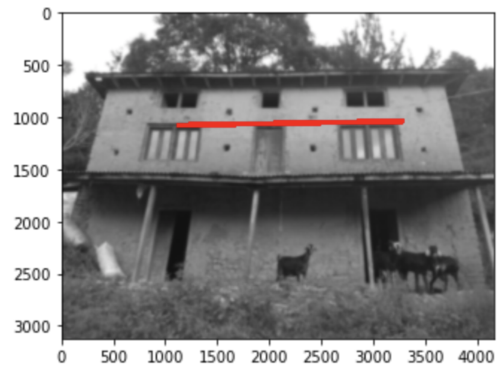


B. Level Detection (Real Images)

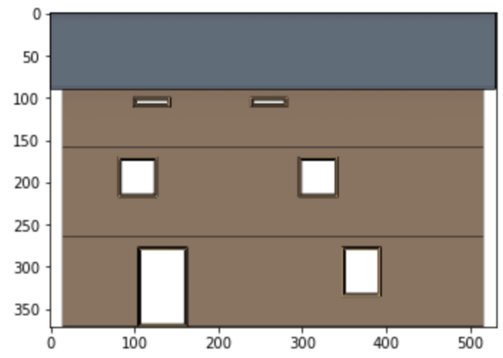
```
number of levels = 3
```



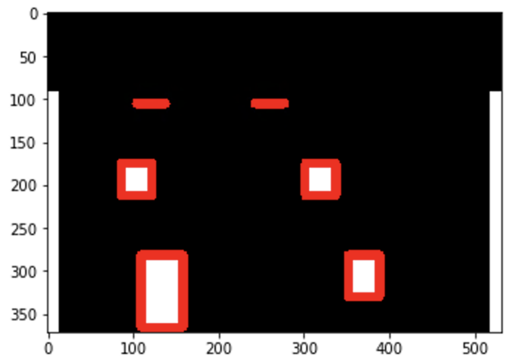
```
number of levels = 2
```

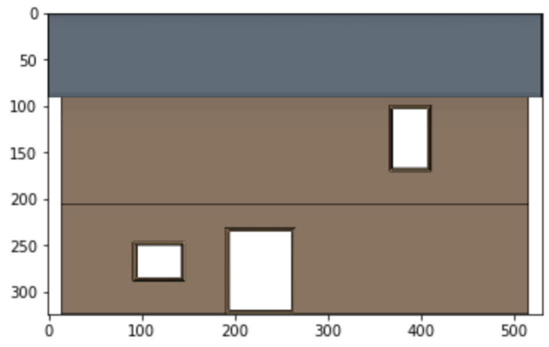


C. Openings Detection (Syntactic Images)

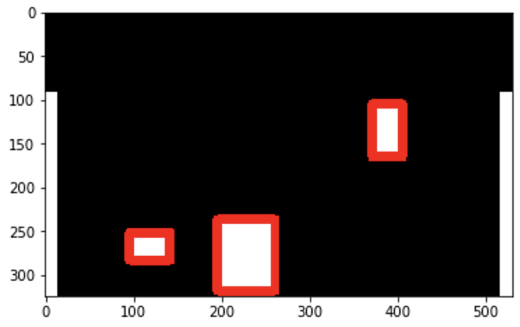


```
number of openings =  6
```



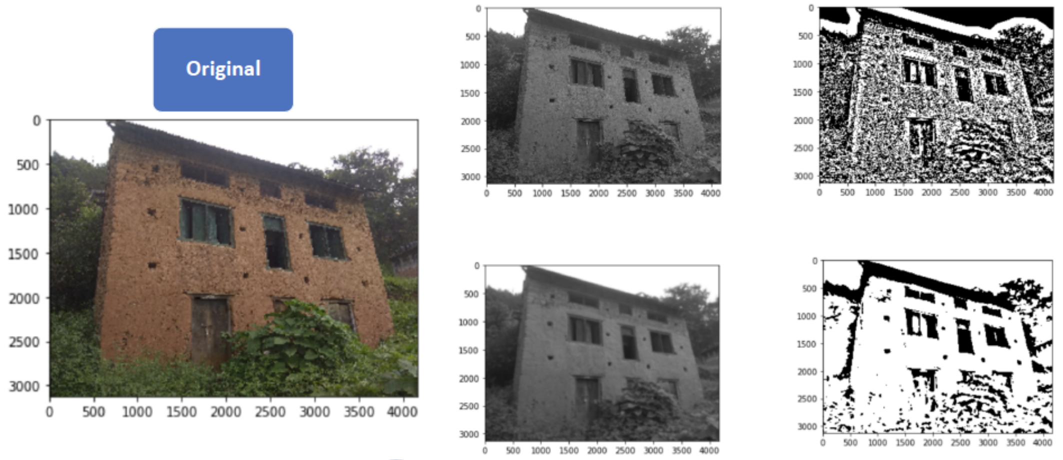


```
number of openings =  3
```



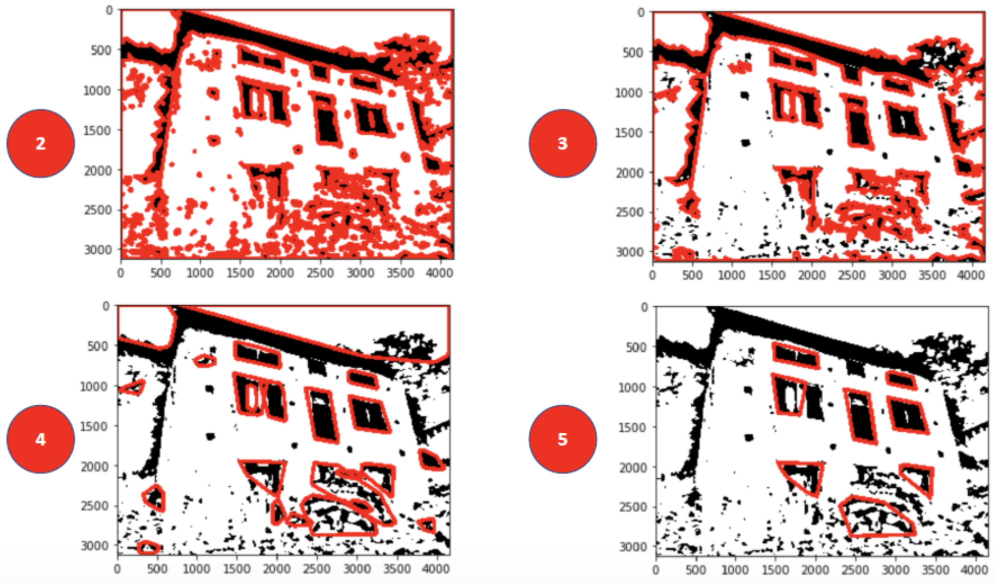
D. Openings Detection (Real Images)

Real Image

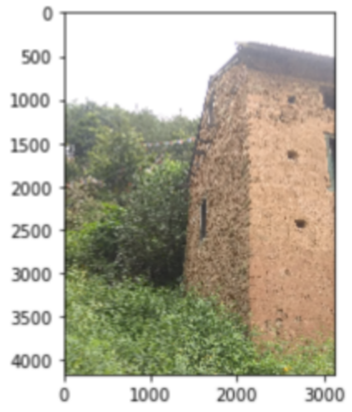


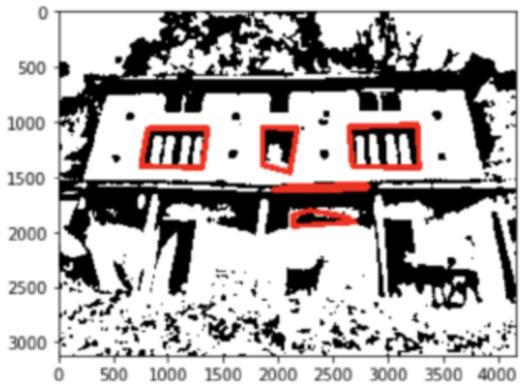
1 **Bilateral Filtering & Adaptive Thresholding**

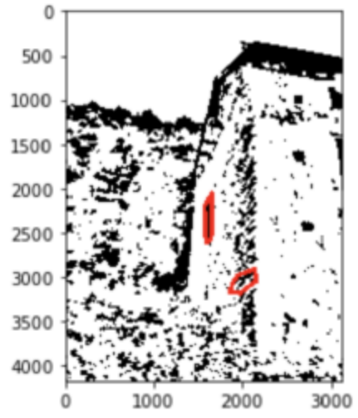
Filtering out bad contours



Other examples:




0
500
1000
1500
2000
2500
3000
3500
4000


0
500
1000
1500
2000
2500
3000
3500
4000