Building Damage Assessment from Satellite Imagery

ENEE 439D Capstone Project

Andrew Orlosky, Alan Cheriyan, and Patrick Marinich
Advisor: Dr. Min Wu

Table of Contents

Signatures and Contributions

Alan Cheriyan - I pledge on my honor that I have not given or received any unauthorized assistance on this report.

Contributions: I tackled a wide array of topics for this project. Primarily, I attempted to get a Mask-RCNN model working. I utilized a GitHub repository, following the ReadMe in order to extend the proper classes and override the necessary functions. However, when I ran into an error during training the model, I was advised to move on from the unofficial repository. After running into this brick wall, I pivoted to preprocessing. I created functions to gather the points that represent each building from the json data, create a mask of the buildings, and make all those masks the same size. As a part of the preprocessing, I also created the code to extract images from our dataset for training and testing. After that preprocessing work, I then added that code to our CNN pipeline to get that model initially working. I experimented with this model, trying different image and filter sizes, train and test splits, and epochs to optimize the CNN. I additionally solely conducted optimizer and loss function testing to find the best pairing. Afterward, I built the SVM model and conducted all the experiments for that architecture. Finally, I contributed significantly to the final report along with my other two team members. All in all, I am proud of our project and how far we have come.

Andrew Orlosky - I pledge on my honor that I have not given or received any unauthorized assistance on this report.

Contributions: For this project, I contributed to many aspects ranging from preprocessing to ML model analysis. Specifically, I spearheaded the effort to balance the distribution of building damage labels in the training and testing datasets. Ultimately, this change in our preprocessing affected all of our model design approaches. In terms of ML analysis, I was highly involved in the building and training the Convolutional Neural Network (CNN). In this part, I wrote code to pull images by disaster type from our dataset and built the sections used in our demo to test a single image using a pre-trained model. Additionally, I led the K-Nearest Neighbor design choice and explained how this approach was beneficial for improving classification performance compared to our baseline CNN architecture. Throughout the course of this project, I led our project management to keep our team organized with documentation and GANTT chart updates such that we stayed on track to meet our project goals. In terms of this report, I contributed mainly to the Realistic Constraints, Design Choices, Technical Analysis, Design Validation, and Conclusion sections. Overall, I was pleased with the results our team achieved in this project and proud of my contributions.

Patrick Marinich - I pledge on my honor that I have not given or received any unauthorized assistance on this report.

Contributions: Throughout the duration of this project I played a large role in multiple different aspects of the final product. At the beginning of the project, I was responsible for initially learning how to interact with and visualize the labels from the dataset. In addition, I wrote the code and spent the time cleaning our dataset from the images with large black splotches in them. For experimentation, I was responsible for

writing the code to visualize our confusion matrices in a presentable and ergonomic way. I was involved in experimentation surrounding our initial testing with the CNN model including experimenting with different architectures and kernel sizes. As we continued development I took the lead on learning and developing the code for the Siamese neural network model as well as running a large amount of experiments with it to see how it performed across different disasters. Towards the end of the project, I spent my time working with the edge detection method of applying a filter to our data before training the model to hopefully see some positive results. As for this report, I was responsible for the Executive Summary, Goals and Overview and Engineering Standards sections as well as subsets of the Technical Analysis and Design Validation sections. Overall, I am proud of the work that I put in this semester and how well that we worked together as a team. I feel that our results and findings are prominent and can be used by future students to bolster their projects and experience in ENEE 439D.

Executive Summary

The xView2 dataset challenge was set up in 2018 to create a competition for AI/ML researchers surrounding the topic of natural disaster relief. The dataset includes satellite images of buildings before and after a natural disaster. The goal of the challenge is to create a model which can both segment and classify the buildings and their damage level. For our capstone project, we focused on the classification aspect of the challenge, and experimented with a multitude of different ML methods to try and create an accurate way to classify the damage level of a building after a natural disaster strikes. After learning how to utilize the xView2 dataset and visualize the data, we began performing experiments with different models. Our experiments mostly involved a singular natural disaster as different types of natural disasters can have vastly different impacts on how damaged buildings look. Throughout this report we will discuss our methods and findings of these experiments that we conducted throughout the semester, including experiments with Convolutional Neural Networks (CNNs), Optimizers, Loss functions, Siamese Modeling, Support Vector Machines (SVM), K-Nearest Neighbors, and Image pre-processing.

I. Introduction

Various natural disasters frequently occur every year. These disasters often tear through civilization whether that be busy cities or rural towns. In order to provide the most effective aid post-disaster, a damage assessment is mandatory. Understanding the magnitude of the damage is an important step to see how much aid is needed and which areas need the most attention. Traditionally, in order to get these assessments, people are sent onsite, manually going around and checking each building. This method is rather crude as it puts people at risk of lingering damages (i.e. damaged power lines) in order to gather information. Furthermore, the act of checking each building one by one is very time consuming and delays the time for the subsequent aid. Ultimately, being able to scan an area and quickly classify the extent of damage in specific areas can enable communities to better respond to natural disasters in a timely manner. Our design takes in satellite images along with the segmented building locations and classifies the damage level that those buildings sustained during the disaster. The goal of our model would be for disaster relief organizations to have the ability to use our model, and hopefully provide valuable insight into where the damage is concentrated.

A potential workflow may include relief workers feeding the data from an overhead camera into a localization model in order to locate all the buildings in each image. Then, the images with located buildings could be used as an input in our model in order to assess the damage level of the buildings.

Our code utilizes machine learning techniques in order to perform damage level classification on those buildings, placing buildings into categories based on the level of damage they accrued. We assess the precision of each model using accuracy and F1 scores. In the xView2 AI Challenge [1], an open challenge to see who's model could identify buildings and rate the amount of damage the best, IBM announced that the best submission achieved an F1 score of 66% for damage classification. We attempted to get similar results as we built our models and performed various experiments throughout the semester.

The final product of our project would be a software model which can classify the damage levels of a building. For our development we used Google Colab as our programming environment as it allows for access to GPU resources for training the model. If our model was to transition into a fully deployed application, then it would be beneficial to have dedicated hardware, specifically GPUs so that the training of the model can be done quickly and efficiently.

II. Goals and Overview

The original proposal for our project was to follow the guidelines of the xView2 challenge and create a model that was capable of segmenting and classifying the damage level of the buildings. [6] As we began development we quickly realized that this project scope would be too large for the time constraint that we had. We pivoted into focusing our efforts onto the classification of the damage level of the buildings. The xView2 dataset provided very strong labeling, and each building was able to be outlined using the labels provided.

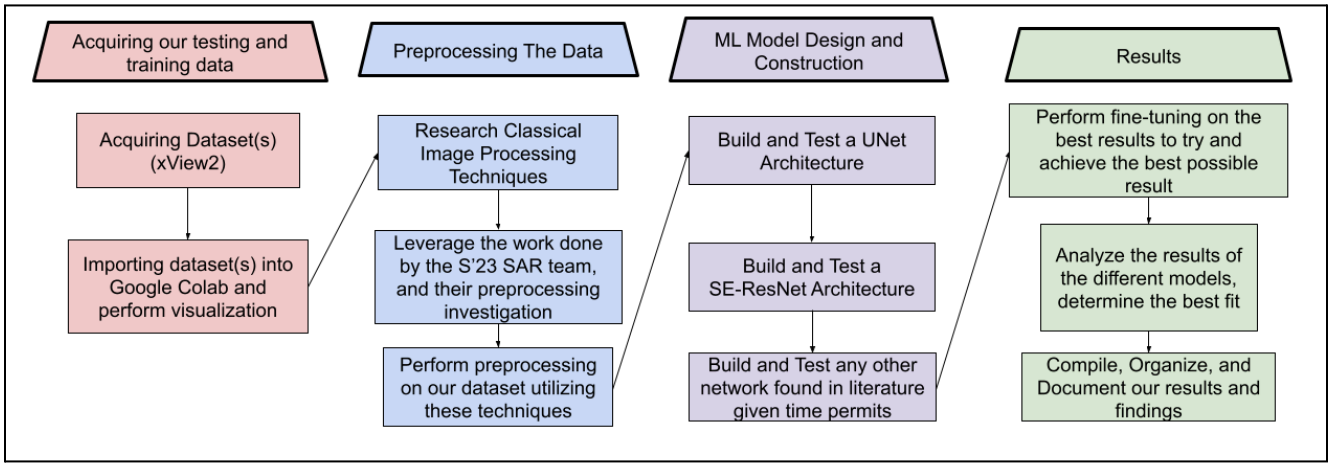

Figure 1 - This was our original workflow proposal, it included some of the finalized workflow aspects such as the dataset visualization and preprocessing ideas, however since we changed the scope of the project once we began development, our final project pipeline contains classification modeling rather than segmentation modeling.

Once our scope was finalized into the classification aspect of the image, we had a clear goal on what we wanted to accomplish. The end goal that we came up with is to take a satellite image as an input with its buildings segmented, and determine the level of damage that the buildings faced after the natural disaster occurred. With this goal in mind, we were able to construct a new pipeline to outline our new scope and goals of our project.

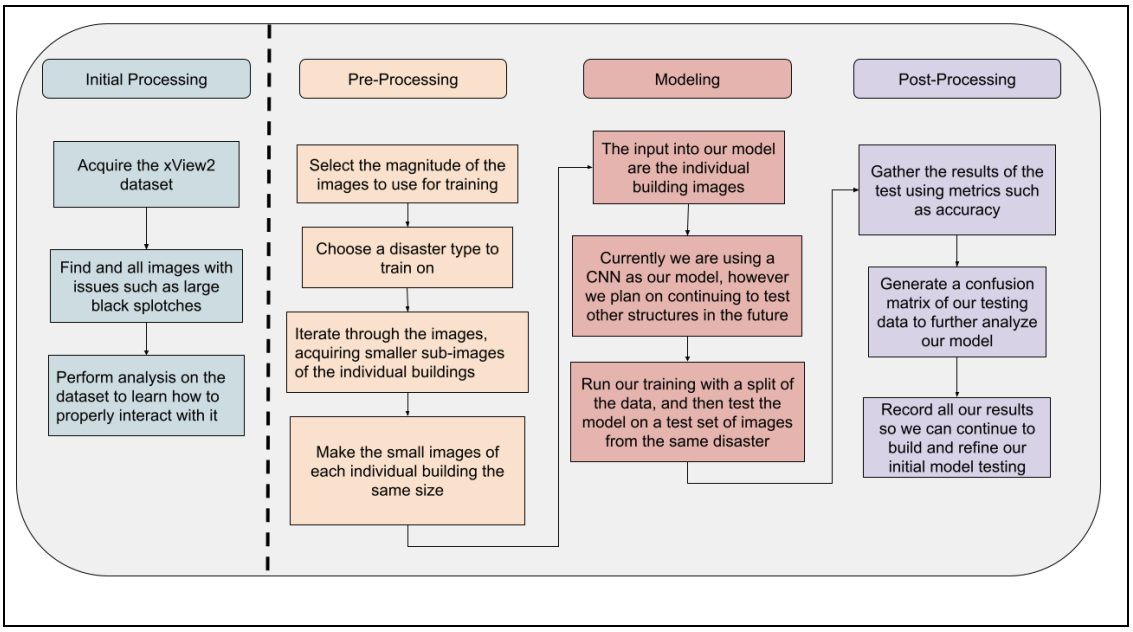

Figure 2 - This is our finalized design pipeline to reflect the final scope and goals of our project. The sections include: Initial Processing of the dataset, pre-processing we have to do on the data, the model itself, and organizing the results that we found. This data pipeline was created for our milestone review and was present in the slideshow we presented in class.

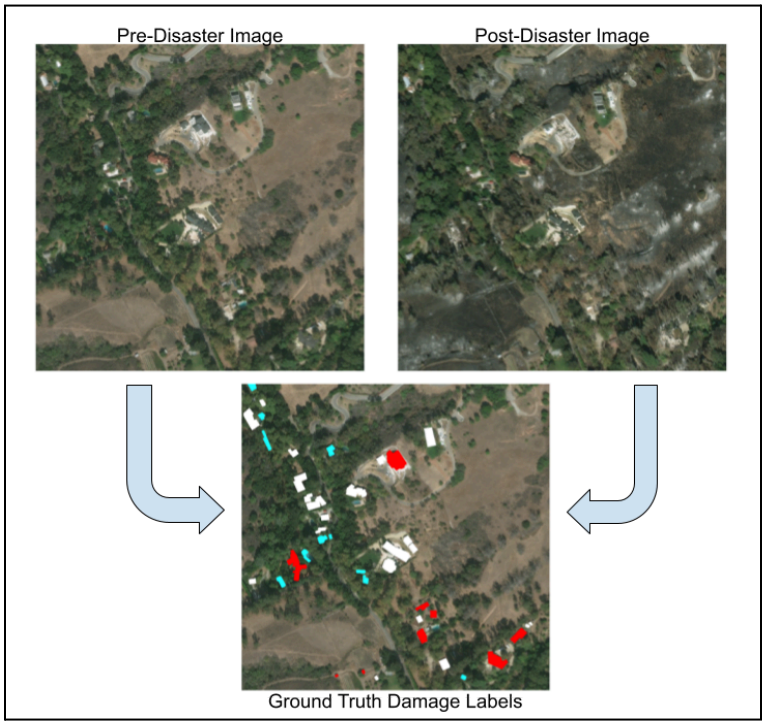

Figure 3 - This is an example of our main goal. All three of these images are from the Southern California disaster dataset. The pre-disaster and post disaster images are of the exact same location before and after the fire happened. The ground Truth damage detection labels are the labels provided by the dataset,

visualized on top of their respective buildings. With red representing the building being destroyed, white representing no damage to the building, and the light blue being buildings that were considered ‘unclassified’ by the label. Our final goal would be to create a model to replicate the ground truth labels as best as possible.

Category	Specific Goal
Data Visualization	Utilize the ground truth labels to be able to overlay the damage levels provided over the image so that there is a user-friendly interface for viewing the results.
Pre-Processing	It is critical based on our assumption to be able to use the labels of the buildings location provided to segment the buildings before inputting them into our model.
Addressing Non-Uniformity of Data	Our dataset is very skewed towards most of the buildings having little damage, therefore we need to find an approach to normalize our dataset in a way such that the count across the damage levels is equal
Modeling	The first place result of the xView2 Challenge achieved an F1 Score of 66% according to our research, since this is the state of the art, our goal would be to achieve a result approaching this value
Post-Processing	A common problem that arises in classification problems is that the model finds a local minimum in the loss function by strictly guessing one category and getting high accuracy, to avoid this we want to analyze our results using a confusion matrix
Methodologies	Due to the nature of the ENEE439D course, it is in our best interest to learn and experiment with different modeling techniques, so it is imperative that we explore more than one method when designing our model.

Figure 4 - A table of specifications representing the goals of our project across six categories

III. Realistic Constraints

Public Health

Our model's goal is to determine the level of damage that occurred to different buildings during a natural disaster. This is directly relevant to public health due to the impacts that large scale natural disasters have on the health and well-being of those affected by them. The primary goal of this model would be to allow organizations who plan on helping after a disaster occurs to focus their resources on the most damaged areas first since these locations are likely to be where the public's well being are the most vulnerable. The success of a model which can accurately classify damages from satellite images can have a direct positive impact on the response to natural disasters.

Environmental Factors

Due to the nature of the data we are working with, natural disasters, environmental factors have been a primary concern within our research thus far. An important distinction that we have made thus far is that different natural disasters have vastly different effects on the levels of damage that a building can suffer. Through analysis on our dataset we have been able to see that typically wildfires lead to very binary damage levels: No-Damage or Destroyed, whereas hurricanes tend to lead to different damage levels across the board. This is something that we have had to consider during our design process as there may not be a one size fits all solution to our damage classification problem.

Global

The dataset we are using is the xView2 Dataset of satellite images of a location taken before a natural disaster happens and afterwards. The dataset however is not evenly distributed across the globe. A vast majority of the disasters present in this dataset are in the western hemisphere, and furthermore they are concentrated in the United States. This may lead to a deployment issue of our model in other parts of the world due to the lack of training information on a local scale. Differences in building construction, background vegetation, and other local environmental factors that are vastly different than in the locations of the majority of the training data may have large consequences on the success of the deployment of a successful model.

Fairness

Within the scope of this project, fairness is critical to the success of every step of the building damage classification process: preprocessing, in-processing, and post-processing. For preprocessing, our team has faced many fairness challenges. With our dataset, we have pre- and

post-disaster satellite images from natural disasters of a variety of types (fires, hurricanes, floods, earthquakes, etc.) each of which from different locations and different orders of magnitude. To address this, we have developed ways to train our model differently for each use case, but in the big picture this dataset is mainly limited to the US and North America, so natural disasters in the EU and Asia are not represented fairly. We have also attempted to achieve during model training in the in-processing phases by training our building damage detection model on a dataset with equally proportional damage classes between no damage, minor damage, major damage, and destroyed buildings. Having an equal building damage variety in training will ensure fairness in our model training and improved classification accuracy across our different classes. In post-processing, we are using a different holdout subset of data for testing our model to preserve fairness in classification. Additionally, we have some pre disaster data that we plan to use to refine our model accuracy and fairly consider all aspects of our dataset.

Economic

On a global scale, there is a vast discrepancy of the preparedness of an area for a given natural disaster. There are even small discrepancies between different locations within the same nation. These levels of preparedness can have a massive impact on how affected an area can be after a disaster occurs. Therefore each and every natural disaster can have a variety of different impacts depending on where in the world the disaster takes place, even if they were of similar magnitude. In addition, different parts of the world may have different definitions of what a ‘Destroyed’ or ‘Minorly Damaged’ building looks like, so making these distinctions depending on the area are critical to the success of a deployed model with the goal of classifying damaged buildings.

IV. Engineering Standards

A designed standard that we followed is an IEEE standard for the development and deployment of AI systems. There are both basic functionalities and secure functionalities standardized in this IEEE Standard. From the IEEE Standards Association, this is the description of the standard: “The MPAI AI Framework (MPAI-AIF) Technical Specification (in the following also called MPAI-AIF V2) specifies the architecture, interfaces, protocols, and Application Programming Interfaces (API) of an AI Framework specially designed for execution of AI-based implementa-tions, but also suitable for mixed AI and traditional data processing workflows.” (https://standards.ieee.org/ieee/3301/11510/). With this standard in mind, we have followed the first five of the basic functionalities as they would be critical if our project was to be deployed on a large scale. The five basic functionalities are:

1. Independence of the Operating System
2. Component-Based modular architecture
3. Interfaces that encapsulate and abstract from the development environment
4. Enabled access to validated components
5. Implementation in software or hardware

Each of these parts of the standard has been addressed during our development cycle. Firstly, our project is independent of the operating system that it is run on, since our development environment for this project has been Google Colab, then a system which can open and run Google Colab, regardless of its OS, would be able to use our system, thus there is no OS dependence there.

Due to the nature of the Google Colab environment our code is very modular within each specific file. Google Colab uses a cell structure where blocks of code can be separated by functionality which allows for a very modular architecture. In addition, at a higher level, our project has separate code files for each of the different models that we designed which further adds to our organization and the modular architecture of our project.

Google Colab is a development environment that uses mainly python as its programming language, and thus for our project all of our code is written in python. Since python is not specific to Google Colab, then we would have the ability to move our code from Google Colab to any other development environment that supports Python. Our code is abstracted away from the initial development environment of Google Colab, but it still needs to be run on a system which supports python.

As of this point in our development, all of our code is runnable through the Google Colab environment, since our project is not in a state where it can be fully deployed we do not have an

interface to interact with our validated components. If we were to continue the development of our project to make it into a deployable model, then it would be critical to create an interface to abstract away our components from the end user so that the end user could have a seamless experience.

Finally, we choose Google Colab as our development environment due to the access it provides to GPU resources. Therefore in its current form, our project is entirely software based, however if it were to become a deployed application, then it would be important to have dedicated hardware for our system to reduce any training times that may need to occur.

V. Alternative Designs and Design Choices

After the preprocessing phase, we focused on narrowing down the criteria for our machine learning problem in order to determine the key approaches for effectively addressing it. Ultimately, the goal of this project was to train a model to predict the damage labels of satellite images post-disaster to provide response teams with advanced insight on where to target for critical natural disaster response. To accomplish this, we evaluated a variety of approaches including segmentation, regression, classification, and traditional ML classifiers. Discussed below, we compared each of these approaches in order to down-select the most appropriate ones for our project.

Segmentation

Primarily, we wanted to perform segmentation similar to the xView2 Challenge. Initially, we thought that we could label each of the buildings as their appropriate damage level classes on the entire image using segmentation. We had previously performed segmentation using the Mask-RCNN on the MS COCO dataset for Lab 5 so we wanted to apply that model to our new dataset. Mask-RCNN builds off a traditional CNN, partitioning an image into multiple segments to locate objects and boundaries, ultimately outputting a mask for each located object. After realizing that Lab 5's MASK_RCNN code would not translate over for the xView2 dataset, we found an unofficial repository implementing Mask-RCNN on GitHub. We followed the instructions in the ReadMe, extending the proper classes and creating the necessary functions. However, we ran into an error when attempting to test the model. Because the repository's code was extremely large and from an unofficial source, we were advised to scrap this work and start anew. Trying to understand the repository would have taken at least a week and even then would not be guaranteed to work. Hence, we decided to move on. We looked into the Unet architecture, another well-known segmentation approach, which utilizes encoder layers that capture features and reduce the spatial resolution of an image. Then, decoder layers will merge the captured information while simultaneously reverting the spatial resolution. Ultimately, we realized that

since we already have the label coordinates for each building, we should focus on damage classification rather than prioritizing creation of segmentation masks with different shapes and sizes for each individual building object.

Regression

Secondly, we considered using a regression model to predict the extent of post-disaster building damages on a continuous scale. To accomplish this, we would have had to apply techniques from Labs 3 and 4 to perform prediction of damage level based on regression or autoregressive modeling. In regression modeling for this case, a model would be trained to predict a numerical value representing the extent of damage for each building, while providing metrics for the statistical significance of the predictors, such as p-value, r^2, and adjusted r^2. However, we determined that this approach would not be as effective given the dataset provided building damages based on a categorical scale of no-damage, minor damage, major damage, destroyed, and unclassified rather than a numeric value corresponding to the damage of the building, which would have suited better for a regression approach.

Classification

Afterward, we pivoted and determined that our project was a classification problem, given we needed to classify buildings as one of the five damage labels. We could utilize these labels, feeding them into the models as training and trying to predict the label of test images. We researched common classification models and initially worked with a Convolutional Neural Network (CNN). In this model, the input images are put through layers of convolutional filters of different sizes with the outputs of the prior layer feeding into the inputs of the subsequent layer. Activation functions, which add bias to outputs so that negative outputs map to zero and positive outputs are maintained, as well as pooling, simplifying the model by downsampling, are other layers utilized in our CNN.

We also experimented with edge detection as a preprocessing technique to aid our CNN. We applied both the Sobel and Canny edge detectors. These edge detectors use different kernels and make the images black and white so that the white highlights the edges in the image and the black shows everything else [18]. We fed these images into the CNN to make it easier for the CNN to classify the damage levels of buildings.

Building off the CNN, we also employed a Siamese Neural Network. This model performs binary classification to see if two images are of the same class or not. You start by feeding in two similar images into identical CNN's. The similarity of the outputs of the two CNN's are then compared using a contrastive loss in order to determine whether or not the two images belong to the same class [13]. This model felt very intuitive to our problem as the

pre-disaster and post-disaster images could be inputted into the identical CNN's. If the model found a high contrastive loss (i.e. the pre- and post-disaster images were deemed to be different), we could determine that the building was damaged after the disaster passed through. Hence, this was another design approach we considered.

Traditional ML Classifiers

In addition to neural networks, we also built traditional machine learning classifiers to compare accuracies against. We settled on two: the Support Vector Machine (SVM) and the K-Nearest Neighbors (KNN) classifier. The SVM maps data to high dimension feature spaces and then categorizes the data points based on the classes [20]. The KNN classifier operates by finding the k nearest neighbors to a given sample and determines which class holds the majority within that set of k neighbors. These were two other approaches that could work with our dataset and seemed viable.

Design Approaches	Segmentation	Regression	Classification	Traditional ML Classifiers
Modeling Techniques	Mask RCNN and UNet	Linear Regression, AR Models	CNN, Edge Detection, and Siamese NN	Support Vector Machines and K-Nearest Neighbor
Dataset Compatibility	4	2	5	5
Ease of Use	1	3	4	5
Accomplishes Problem Goals	3	2	5	5
Training Efficiency and Performance	4	3	5	5
Advisor Recommendations	3	3	4	5
Total	15	16	**23**	**25**

Figure 5: In this chart above, we used a Pugh Matrix to determine which of the design choices we should focus on over the course of our project. Each of the options were rated on a scale of 1-5 from least to most successful for satisfying the appropriate criteria.

Design Choices

In our case, since we wanted to predict both whether buildings are damaged and the damage level (which involves multiple discrete categories), we focused on using multi-label classification and traditional ML classifiers. For this approach, we will train ML models to predict multiple labels (e.g., whether damaged or not, and if damaged, what level of damage) for each image rather than a single numerical value. Rather than focusing on building detection or localization, we prioritized learning the damage labels from each building directly to better optimize performance. Based on the criteria for our ML problem and the results from our Pugh Matrix, we determined that these traditional classification approaches will help us best accomplish our project goal.

VI. Technical Analysis for System and Subsystems

Dataset

The xView2 Dataset is a vast dataset of satellite images that were taken before and after various natural disasters.[2] According to the xView2 website and paper [3], there are 850,736 buildings throughout the entirety of the dataset, across 17 different disaster types. Due to the massive size of the entirety of the dataset, we decided to use 12GB of the approximately 51GB of data available, to save resources, downloading time and pre-processing time. We determined that the 12GB encompasses the dataset in a meaningful way without too much loss. Our subset of the data contains 10 of the 17 disasters that are available, and span across the different types of disasters that the dataset contains.

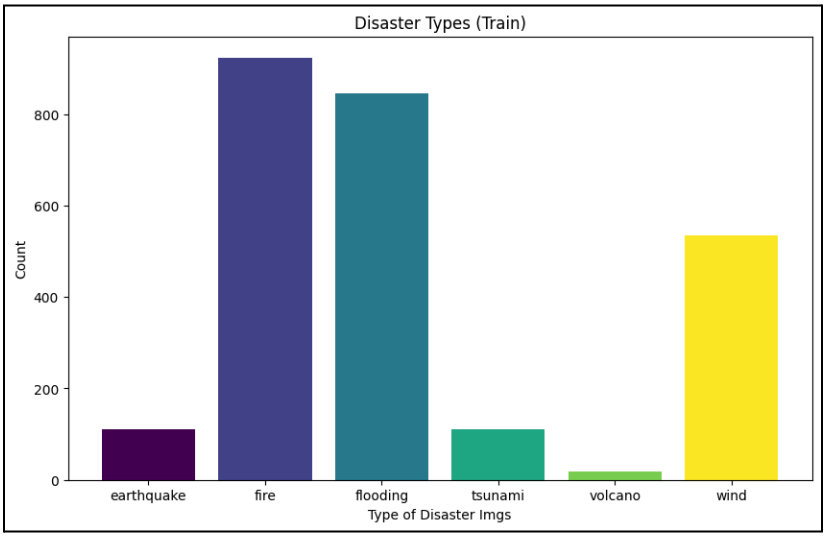

Figure 6 - This is a breakdown of the image count of each of the different disaster types that we had access to in our subset of the dataset. We account for all of the different disaster types,

however the volcano portion is definitely smaller than we initially anticipated. Note this graph is of the image count and not the building count, so even though the earthquake and tsunami image count is low, there were plenty of buildings for our model to train on for these categories, since these images were dense with buildings.

The xView2 Dataset classified damaged buildings into one of five different categories, no damage, minor damage, major damage, destroyed, and unclassified [4]. For our modeling purposes we decided to ignore the unclassified buildings and focus on the four classes of damage. The labeling scheme of the ground truths was very strong as it provided the locations for the buildings and their respective damage level. This allowed us to be able to overlay the damage level of each building over the satellite image as seen in *Figure 3.* [9]

There were some challenges that we had to work through with the dataset that are important to mention. The largest of these issues was that some of the images had missing information, large black spots, that covered a large portion of the images potentially blocking out or cutting off buildings. To fix this issue we had to do some initial processing on the dataset. We wrote a program to iterate through each of the images checking for these large black spots by using a kernel of size 10x10 looking for large spots with no data in them (zeros), if the image was found to have this, we deleted it and its corresponding pre-disaster or post-disaster image. Once this process was complete our dataset only contained images that were not missing information, and it was ready to be used.

Given that the images were cleaned of any that were missing information, the data was ready to be used in the next set of our pre-processing, which included gathering the images by the disaster type. One of our goals was to see how the effect of a different disaster affected the results of our models. To do this we needed a way to sort through the images and gather the ones that correspond to the same disaster and use those for the training and testing of our model. Another positive aspect of the xView2 dataset is that the images have well constructed path names which can be parsed to determine which disaster they come from. When beginning a training session of one of our models, the first thing that our program does when provided a disaster name, is to sort through and gather only the images from that disaster, so that the images of the specified disaster are isolated for training.

The final step of pre-processing of our data before beginning the training aspect of our models is to segment the buildings in the image. We defined our model structure to take in the image of an individual building rather than the whole image, then once all of the buildings in the image have been processed the results can be overlaid on top of the original satellite image. To segment each of the buildings we use the labels provided by xView2 to get the location of the building in the image, then we cut out and resize the building to be a standard size [7]. If the building image needed to be made larger then we zero padded the image to the proper size, if it

needed to be smaller we used the CV2 resize method. Once all of our buildings images have been created and resized to be the same, we then create our train and test splits of the data, one of our goals with this is to normalize across the damage level categories so that there are an equal number of datapoints for each class [8]. Therefore we counted up all of the different instances of each damage class, then took the minimum of these and used that many images of each damage classification, we then finally used a standard 80%-20% split of the data for our training and testing sets.

CNN

For CNN, we started with a baseline architecture based on the one used in Lab 1, which is shown in Figure X below. As input, we took a list of post-disaster images from a specific disaster type (hurricane, earthquake, fire, flood, etc.). Using the building json labels, we selected the preprocessed building images of size 150x150 as input. The CNN had 3 convolutional layers of decreasing sizes, followed by a max pooling layer, before employing dropout, flatten, and activation layers in order to get a final label prediction.

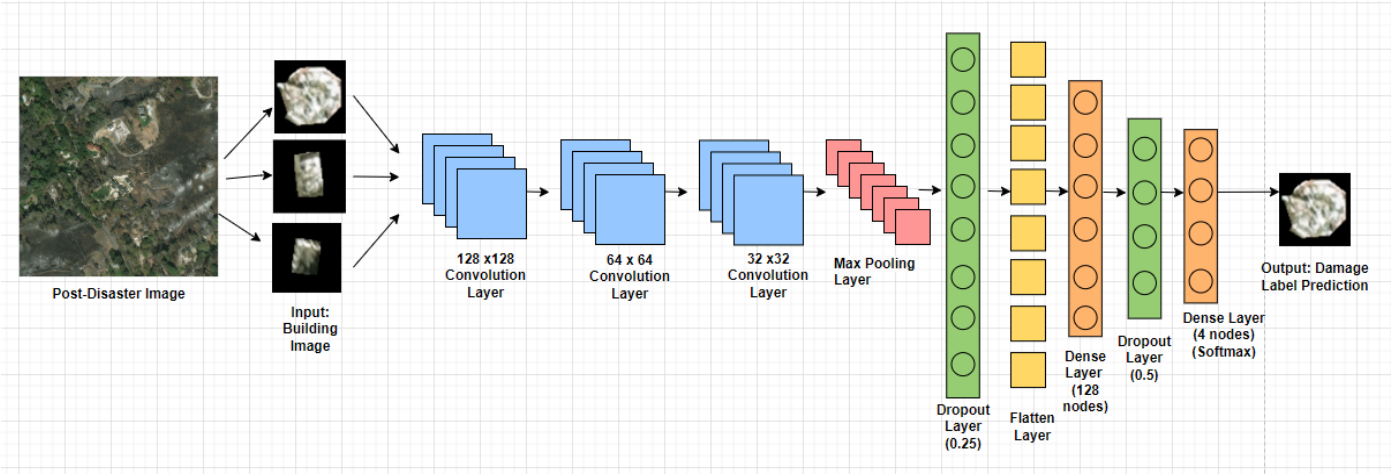

Figure 7: This figure shows the baseline CNN Architecture used in our training with 3 convolutional layers, max pooling, dropout layers, flatten layers, and dense layers to predict building damage levels for individual buildings.

Our initial baseline results had a wide variety of test accuracies across the different disasters, ranging from 30% to above 80%. Because of these unexpected results, we used a Confusion Matrix to evaluate the prediction distribution across the different damage labels. The goal of the confusion matrix is to show how the model is classifying the images with respect to the ground truth labels. The goal is to see the highest numbers be placed across the diagonal from top left to bottom right as those are the boxes where the ground truth matches the model prediction. [10,11,12] The left matrix shows the issue of the model strictly guessing one category (non-damage), whereas the right shows a more diverse prediction pattern. Ideally, we would like

to have 100% accuracy on prediction from top left to bottom right for all four damage level classes.

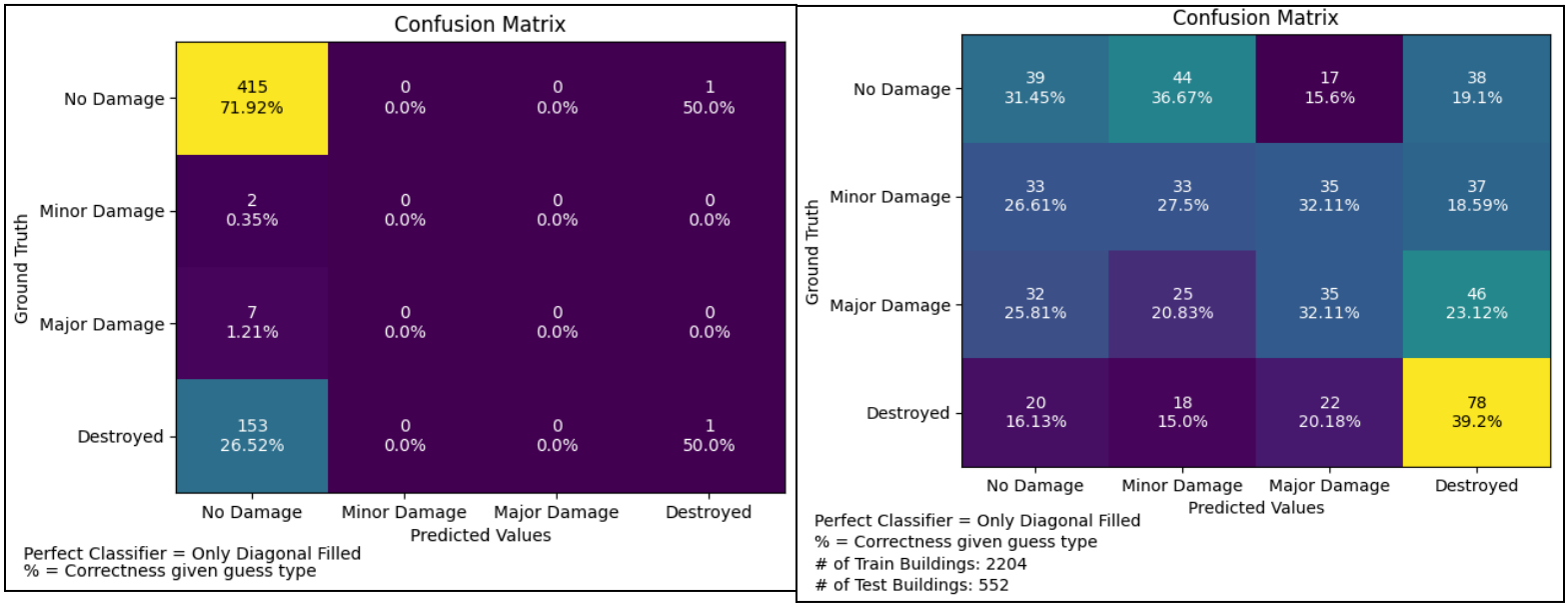

Figure 8 - These are two of the confusion matrices generated in our CNN testing. In the left image, we see the CNN only learning no damage, which led our team to redesign our training and testing data with an equal distribution by damage level, shown in the right image with more improved and distribution predictions.

- Left: Southern California Fire with larger 7x7 and 5x5 kernel sizes
- Right: Hurricane Matthew equal distribution of building damage levels

To address this issue, we modified the input to have a more balanced distribution of each building damage level in training and testing, such that the CNN was not biased in training toward one of the classes. In addition to these redesign approaches, we customized the loss function, optimizers, and kernel sizes for different test cases to evaluate how they each affected our results against the baseline. In our testing, we used Adam, Stochastic Gradient Descent (SGD), and Adagrad loss functions, compared Huber, Mean Absolute Error (MAE), LogCosh, and sparse categorical cross entropy, and tested various kernel sizes including 3x3, 5x5, and 7x7. Using the baseline CNN architecture, we ultimately found that the Adagrad or Adam with sparse categorical cross entropy had the highest testing accuracy with 84.15% and 84.95% on a building sample size of largely no damage and destroyed. We also learned that a smaller kernel size had a higher performance accuracy than larger kernels, since it can better learn the features of a damaged building in convolution.

Lastly, we also trained a modified version of the CNN with larger convolutional layers on building images that were preprocessed with Canny and Sobel Edge Detection. Using Canny detection seemed to provide the best results for the model as the sobel model visually seemed to incorporate much more noise into the output image [19]. Using edge detection, although cursing the accuracy to go down, made our confusion matrices start to take the form that we want with the entries along the diagonal being highlighted more so than the other entries.

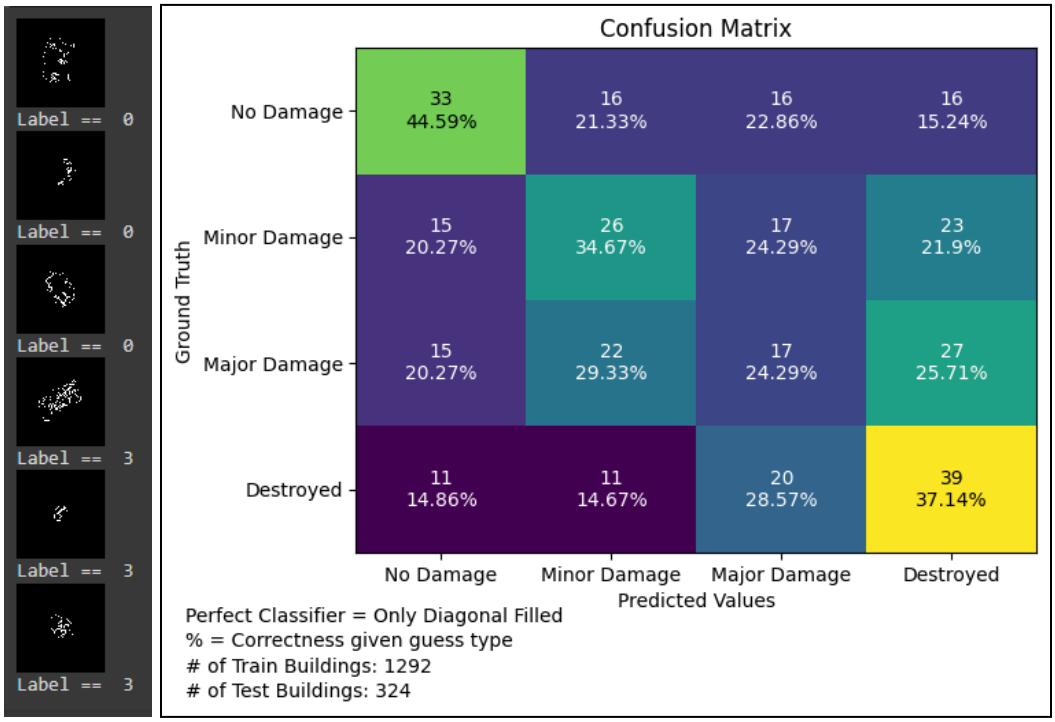

Figure 9 - This figure shows the results of one of the edge detection experiments that we did. The left image is what the buildings look like with the canny edge detection filter over them. It is tough to distinguish the difference between the no-damage (0) and destroyed (3) with just looking at it, but the no-damaged images seem to be on average more sparse than the destroyed ones. The confusion matrix on the right illustrates one of our tests, it achieved an accuracy score of 35% which, although low, is 10 percentage points higher than randomly guessing so learning is occurring.

Siamese Neural Network

The use of a Siamese Neural Network seemed intuitive given the dataset that we were working with. The xView2 dataset provided both a pre disaster and post disaster image of the buildings, so contrasting those images may lead to some interesting results [5]. The idea is that the buildings in the preimage belong to the No-Damage class since the disaster hasn't happened yet, and thus if the model determined that the post-disaster image is of the same class, then the buildings label should be no-damage, where as if the model determined that they were different, then the label should be one of the three other damage labels. For this implementation the data

was sorted into two classes, no-damage and damage, where the damage category contained the three damage classes of minor, major and destroyed. [15]

The Siamese Network that we performed our experiments on was designed with CNNs with 13 layers, and a contrastive loss section of 2 layers. The CNN structure is an augmented version of our CNN testing, by experimenting with the size and number of layers, the 13 layer architecture netted the most promising results. There are 4 convolutional layers of increasing size (64,128,256,512) with max-pooling layers in between each. Then what follows is a flatten layer and 2 dense layers one with 1024 nodes and the final with 144 nodes, the remaining two layers are dropout layers in between the dense layers. Both of the sister CNNs had this structure as the goal is to determine the similarity between the two inputted images.

The contrastive loss layers contain a euclidean distance layer, which computes the distance between the two output vectors. Finally the last layer outputs either a 1 or a 0 indicating if the inputs are in the same class or a different class. [16,17]

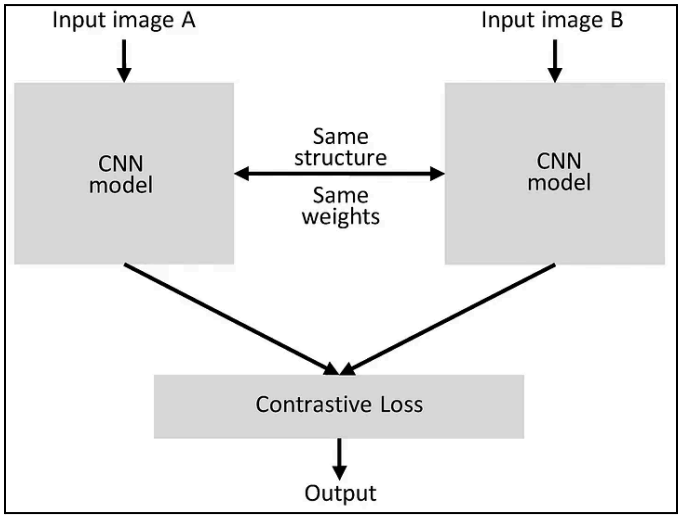

Figure 10 - This is a block diagram of a generic Siamese Neural Network. It combines two identical CNNs together and compares their outputs to determine if the pair of objects are from the same class as each other. [23]

The initial results for the Siamese Model were skewed in one category or the other. It seemed that initially the model would default to guessing one of the two categories to try and achieve the highest accuracy possible. This local minimum in the loss occurred since the data was not evenly distributed between the two potential classes, and thus the model was able to achieve an accuracy score of 57% by guessing only one of the categories.[14] This effect was mitigated by normalizing the classes and increasing the size of the model. Once these changes were applied, then the model achieved a new accuracy score of 58% with a slightly better guessing distribution.

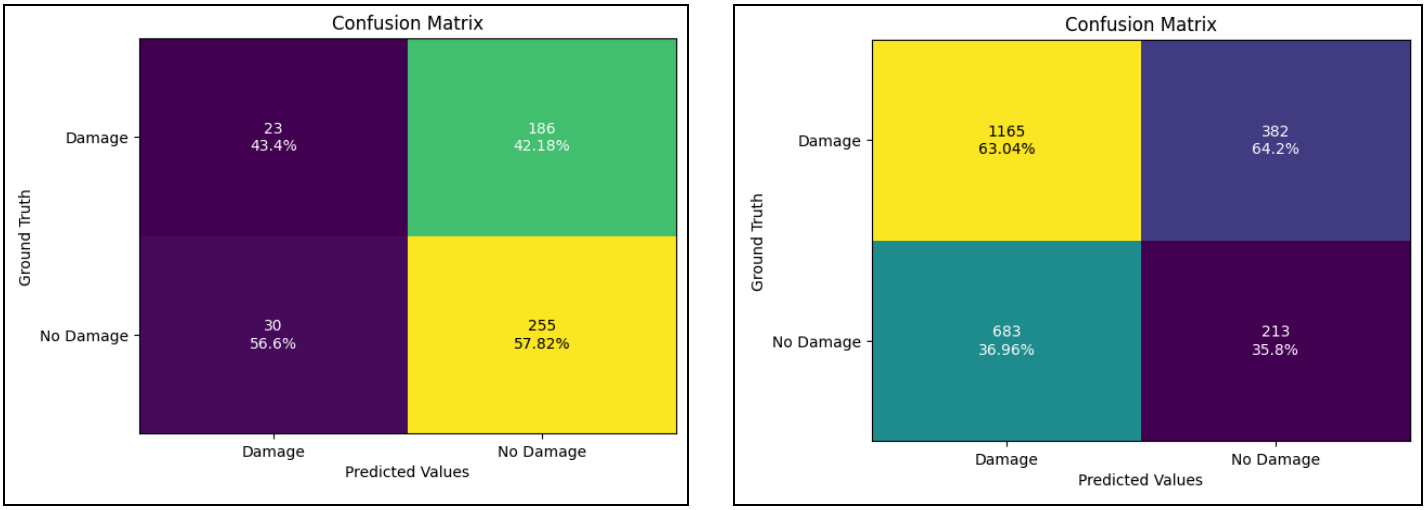

Figure 11 - The left confusion matrix is showing the initial results of the siamese model, where it was predicting the no-damage category for almost every data pair, and the right image is the confusion matrix with our improvement, it is still very biased towards one category but better then the initial group.

Support Vector Machines (SVM)

We built two different SVM's: an RBF SVM and a Polynomial SVM. Both the RBF and Polynomial kernels utilize different functions to create curved decision boundaries around the different classes [21]. We used the standard parameters for both kernels. The RBF SVM uses a gamma of 0.5 and a C of 0.1 and the Polynomial SVM uses a degree of 3 and a C of 1. Before inputting the data into SVM, we first flattened the 150x150x3 images and then performed Principle Component Analysis (PCA) to capture the most information in the fewest data points. Depending on the disaster we tested on, the number of PCA components varied. After training the models, we then predicted the classes of the test set and calculated the resulting accuracies and F1 scores. We performed both regular classification (classifying buildings into one of the four previously mentioned categories) and binary classification (classifying buildings as either no damage or some damage). In regards to regular classification, I achieved a wide range of accuracies, from 25.00% to 93.53%, depending on the disaster and specific kernel used. I quickly found out that the RBF SVM predominantly guessed one class the entire time. The below figure shows the RBF SVM's predictions on the Hurricane Matthew images, the most diverse disaster in the dataset. Despite the evenness of the classes, the classifier still guessed "no damage" every time.

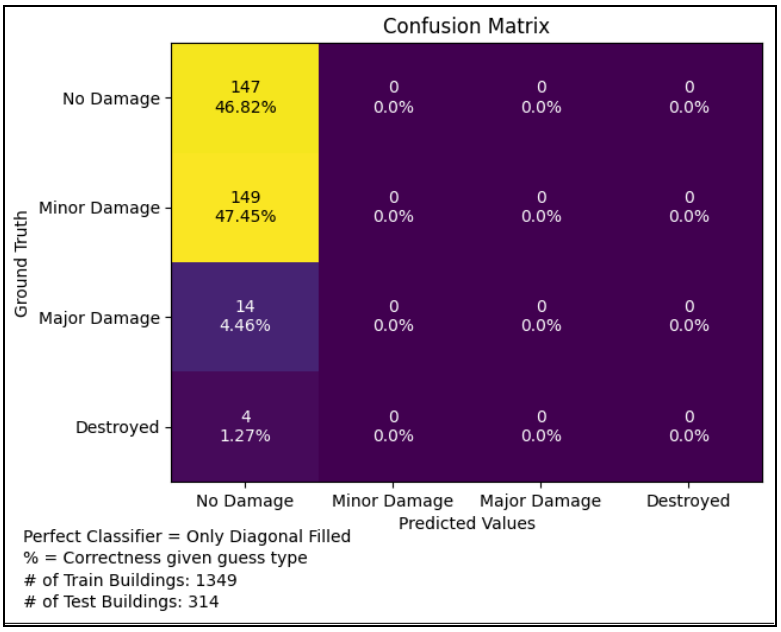

Figure 12 - The above confusion matrix illustrates the results of the RBF SVM on the Hurricane Matthew disaster images. The model guessed "no damage" every time, attaining an accuracy of 46.82% and an F1 score of 29.86%. As a result, I mainly focused my experiments on the Polynomial SVM, editing characteristics of the training sets to improve the accuracy of the model.

K-Nearest Neighbors (KNN)

For the KNN classifier, we used the sklearn package iteratively to determine how the number of neighbors affected the accuracy of building damage label classification [22]. Using the Hurricane Matthew disaster with a balanced distribution of building damage levels, it was determined that as the number of neighbors increases, the prediction accuracy of labels when compared with the ground truth increases. This is extremely promising compared to prior studies because the accuracy reaches around 62% and the F1 score is approaching the maximum score from the first place team in the XView2 challenge of 0.66.

KNN Results - Hurricane Matthew		
# of neighbors	Accuracies	F1 Score
1	0.534913	0.540732
2	0.473815	0.479983
3	0.568579	0.559998
4	0.541147	0.540513
5	0.592269	0.575504
6	0.569825	0.560425
7	0.597257	0.573296
8	0.583541	0.565478
9	0.597257	0.569662
10	0.607232	0.580036
11	0.597257	0.561506
12	0.604738	0.57143
13	0.610973	0.570528
14	0.618454	0.577029
15	0.619701	0.573377

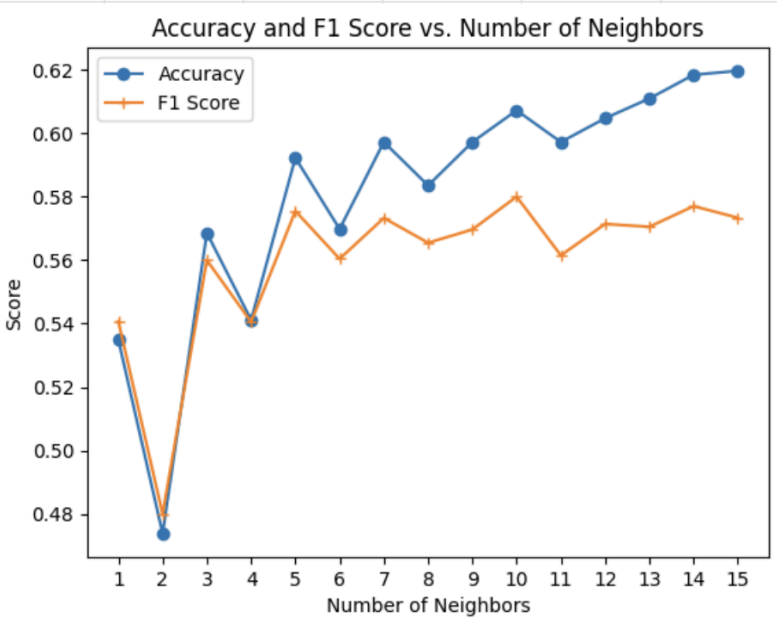

Figure 13: In the image above, the KNN testing results for Hurricane Matthew are presented. The increasing accuracy and F1 score as we raise the number of neighbors is a general trend exhibited when testing other disasters.

VII. Design Validation for System and Subsystems

Model Training

We tested a variety of Machine Learning techniques, throughout the course of the semester which gave us a wide range of different results. Across all of the techniques that we experimented with, they each had their own strengths and weaknesses. We trained our models on multiple different disaster types to see how they changed across the different types, but performed a majority of our testing using the Hurricane Matthew images and the SoCal Fire images since they provided different insights. The Hurricane Mattthew subset of the images had a wide range of data across all four of the damage levels which was very beneficial for the normalization process we developed. The SoCal Fire subset were the opposite as almost all of the buildings fell into either the no damage or destroyed category. These two contrasting datasets provided a good coverage of what the dataset had to offer. In addition, towards the end of the experimentation period we experimented with running tests across the broader category of disasters such as over all of the different fires or different hurricanes.

Performance Metrics

Accuracy is a standard metric, calculated as the correct predictions divided by the total predictions. While this is a good measure of our model, it can be misleading. For instance, an

accuracy of 90% may seem promising, but if the model achieves this accuracy solely guessing one class the entire time, then the model is not as good as the accuracy may make it seem. It is good to evaluate a model with an F1 score in case of such imbalanced data as it combines the precision and recall scores to gain a better understanding of model performance.

Model Evaluation

Accuracies	CNN	CNN with Edge Detection	Siamese NN	SVM	KNN Classifier
Hurricane Matthew	55.97%	33.24%	58.30%	54.40%	57.8%
Guatemala Volcano	91.94%	91.79%	32.00%	92.81%	96.89%
Palu Tsunami	83.48%	40.24%	77.5%	62.30%	86.23%
SoCal Fire	84.95%	66.00%	42.33%	82.13%	76.98%

Figure 14 - This table displays the different models' test accuracies against different disasters. This is just a subset of our results, the full list of our experiments and results can be found here: Building Damage Level Classification Results

The above table shows how our different models performed against different disaster types. We included test accuracies against a hurricane, volcano, earthquake (Palu Tsunami), and fire. Different disasters had different class splits. For instance, Hurricane Matthew was one of the most diverse disasters, having a decent amount of all four classes. This allowed us to perform an equal class split on training. In other words, for this disaster, we could train on an equal amount of images from all four classes. The model, hence, learned about all four classes well. A lot of the data for the volcanoes, earthquakes, and fires on the other hand contained buildings that were predominantly either no damage or destroyed. As a result, models that trained on these disasters could not utilize the equal class split because there simply were not enough images in some of the classes. Furthermore, this characteristic influenced the models to learn the two mentioned classes much better since they dominated the datasets. Because the models were mainly predicting between two classes, the Guatemala Volcano, Palu Tsunami, and SoCal Fire disasters had higher average test accuracies despite the model utilized. However, even these disasters which mainly consisted of two classes were often dominated by buildings classified as no damage. As a result, some of our models only guessed the most common class the entire time. This just goes to show that we could improve our models going forward. Ultimately, we would

argue that the KNN Classifier performed the best as it was on the higher end of the range of accuracies for each disaster.

VIII. Test Plan and Overall Performance Achieved

It is hard to explain our test plan without detailing the entire process our models go through. Primarily, we gathered the data by disaster. Depending on whether or not we had enough images from all four classes, we tailored the data so that there would be an even number of images from every class. After this optional step, we trained the model on 80% of the inputted images and tested on the remaining 20%. Hence, we varied the total number of images and the disasters in our test plan and experiments in order to assess each model. When obtaining our results, every model outputted a test accuracy based on how well they predicted the classes of the test split. Furthermore, our SVM and KNN Classifier also produced F1 scores to additionally gauge these models.

To evaluate our overall performance, we compared our results against the probability of randomly guessing as well as compared to the original state-of-the-art IBM model. Our models exceeded randomly guessing on Hurricane Matthew, which allowed for equal class splits as mentioned earlier. Randomly guessing would have gotten around 25% accuracy whereas our models averaged a little over 50% accuracy for this disaster. As the table in the previous section shows, even on disasters mainly containing two classes, our models usually outperformed the statistical probability of guessing between two classes, 50%. The IBM model achieved an F1 score of 66% when classifying buildings. The SVM had a wide range of F1 scores, from 29% to 90%, depending on the disaster type and other model specifics. Averaging the F1 scores for our best runs across different disasters, the SVM had an average F1 score of 61%. The KNN Classifier's F1 scores were a lot more concentrated. Ultimately, this classifier had an F1 score of 58% against the Hurricane Matthew disaster which we tested on the most. These results are close to the state-of-the-art IBM model and illustrate how our models have performed well overall.

In later experiments, we trained and tested on all the disasters of a specific type. For example, we ran the SVM against a subset of all hurricane data. When doing so, we found that this could result in slight improvements to the model. This goes to show how we can still improve our results given more time.

IX. Project Planning and Management

Over the course of this 10-week project, we used multiple project management techniques to keep our work organized and on track with the project goals. For project planning, we used a GANTT chart to divide our project tasks into 5 distinct phases, each spanning roughly a two week agile sprint to accomplish specific goals. Highlighted in distinct sections of the GANTT chart, the phases we focused on were dataset acquisition and analysis, design choices exploration, classification modeling and tuning, classical ML methods for training and refinement, results wrap-up, documentation, and further work ideas. In terms of responsibilities, we used the GANTT chart to specify individual tasks, identifying which team member or all team members owned the task. In addition, we kept a running weekly log of team member responsibilities, accomplishments, and blockers. This format was very effective because at the end of each week we communicated with each other on what we needed to complete by the end of the phase. Additionally, we used this time to ask for advice on how to approach our blockers and provided feedback on one another's progress with tasks. In the long-run, this project management approach kept us compliant with individual tasks and on track with our project deadlines. Below, I provided the GANTT chart and weekly project log documenting our progress with individual and overall team responsibilities.

CAPSTONE PROJECT GANTT CHART

PROJECT TITLE	Buiding Damage Detection Assessment	PROJECT NAME	
COURSE	ENEE 439D	TEAM MEMBERS	

TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE	WEEK 1				
							M	T	W	T	F
Phase 1	Dataset Acquisition and Analysis	(specifiy All or team member)	3/8/2024	3/24/2024	1-2 weeks	100%					
1.1	Identify Project Topic	All	3/8/2024	3/9/2024	-	100%					
1.2	Project Proposal	All	3/8/2024	3/11/2024	-	100%					
1.3	Project Proposal Revision	All	3/12/2024	3/16/2024	-	100%					
1.4	Acquire SAR Building Dataset and Intial Analysis	All	3/12/2024	3/15/2024	-	100%					
1.5	Preprocessing with Xview Dataset	All	3/15/2024	3/21/2024	-	100%					

Figure 15: This is a brief section of the GANTT which our team used to track progress on weekly tasks and access our long-term project goals using phases for agile sprint planning. For more information on our full project development, see the GANTT chart and our weekly progress log below:

Full GANTT Chart

Weekly Project Log

X. Conclusions

In summary, along with preprocessing of the post-disaster satellite images, we designed a few key solutions to perform building damage assessment: classification with a CNN, Siamese Neural Networks, and traditional ML classifiers (SVM and KNN). Once we determined that building damage assessment was mainly a classification problem, our team successfully performed a comparative analysis of classification techniques each of which had unique features and results demonstrated by varying accuracies, F1 scores, and Confusion Matrices.

Final Design Choice

Ultimately, we determined that the traditional classifiers were the most effective approaches for building damage assessment, but our performance could be improved in all of them by further addressing shortcomings in our training process. More specifically, the KNN had the highest classification accuracy across multiple different disaster types (hurricane, volcano, tsunami, and fire). Additionally, as we increased the number of neighbors in our testing, we were able to achieve results close to the maximum F1 score of 0.66 as tested by IBM [1]. Throughout this project, we learned many lessons that helped us successfully overcome project barriers, which are discussed further below.

In general, our models were most successful with an equal distribution of building damage levels, but we recognize that there is still potential for improvement of testing accuracy and applicability across more disasters. Ultimately, we hope to build a model to perform above 90% accuracy across all four categories of damage level as a long-term project goal. Overall, building damage assessment has many key real-world applications for natural disaster recovery response and has many areas of growth. In terms of future possibilities, we address multiple avenues for further work to expand upon our final results in various applications.

Lessons Learned

Some key lessons that our team learned from this experience were the value of organization and communication. None of our achievements would have been possible had we not been organized with project goals from the start. Using a GANTT chart with pre-planned agile sprint phases, we were able to stay focused and accomplish our goal to have multiple ML methods for performance comparison. Also, throughout the project, there were many points where it was helpful to re-evaluate our progress as a team in our weekly recaps. With strong communication, we were able to avoid stalling progress and easily pivot from issues to other solutions that served more successful at performing building damage classification.

In addition, we learned the value of preparation and flexibility throughout the course of this project. In terms of preparation, it was critical that we evaluated our design choices at the beginning of the project. This helped us save time and focus on solving a classification problem rather than segmentation or regression, which were less aligned with our project goals. Throughout the course of this project, we were fortunate to have a set of mentors with the ENEE 439D teaching staff who provided us with guidance and advice on how to overcome problems. One example of this occurred after the milestone review in which we shifted to focus on traditional ML classifiers to compare with our baseline CNN testing. This was extremely beneficial and helped us extend our application to more traditional methods, which fortunately had higher performance in general.

Further Work

One key avenue for further work of this project is to refine training of our models to increase the building damage classification accuracies. Due to time, we were limited in this effort but accomplished some improvements over a baseline CNN. Additionally, further work could be done to apply the segmentation solutions before the classification, such that a satellite image could have buildings localized and then applied to our models for damage level classification. More so, we only used a subset of the xView2 dataset, so future work could gather more data across different disasters to address the global and environmental realistic constraints. With this, more post-processing work could be performed to address the public health constraints by providing strategic plans for natural disaster response teams. Ultimately, this project was very successful and serves as a baseline to provide further studies to refine building damage assessment and expand to new applications across other infrastructure related disasters.

References

General References:

[1] Alstad, C. (2020, April 1). The xView2 AI Challenge. IBM Blog. https://www.ibm.com/blog/the-xview2-ai-challenge/.

[2] Gerard, S., Borne-Pons, P., & Sullivan, J. (2024). A simple, strong baseline for building damage detection on the XBD dataset. *Arxiv*. https://arxiv.org/pdf/2401.17271

[3] Gupta, R., Goodman, B., Patel, N., Hosfelt, R., Sajeev, S., Heim, E. T., Doshi, J., Lucas, K., Choset, H., & Gaston, M. E. (2019). Creating xBD: A Dataset for Assessing Building Damage from Satellite Imagery. *Arxiv*, 10–17. https://doi.org/10.1184/r1/8135576.v1

[4] Hosfelt, R., & Gupta, R. (2019). xView2_baseline [Computer software]. DIUx-xView. https://github.com/DIUx-xView/xView2_baseline

[5] May, S., Dupuis, A., Lagrange, A., De Vieilleville, F., & Fernandez-Martin, C. (2022). BUILDING DAMAGE ASSESSMENT WITH DEEP LEARNING. ˜*the œInternational Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences/International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLIII-B3-2022*, 1133–1138. https://doi.org/10.5194/isprs-archives-xliii-b3-2022-1133-2022

[6] xView2. (n.d.). Computer vision for building damage assessment. xView2: Assess building damage. Retrieved from https://xview2.org/

References for Preprocessing:

[7] *Cropping Concave polygon from Image using Opencv python*. (n.d.). Stack Overflow. https://stackoverflow.com/questions/48301186/cropping-concave-polygon-from-image-using-opencv-python

[8] GeeksforGeeks. (2023, January 18). *Draw a filled polygon using the OpenCV function fillPoly()*. GeeksforGeeks. https://www.geeksforgeeks.org/draw-a-filled-polygon-using-the-opencv-function-fillpoly

[9] Lezwon. (2019, December 31). *XView2 Challenge*. Kaggle. https://www.kaggle.com/code/lezwon/xview2-challenge#kln-47

References for Confusion Matrix Design

[10] *Matplotlib.pyplot.pcolormesh — Matplotlib 3.8.4 documentation*. (n.d.). https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pcolormesh.html

[11] *Creating annotated heatmaps — Matplotlib 3.8.4 documentation*. (n.d.). https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html#sphx-glr-gallery-images-contours-and-fields-image-annotated-heatmap-py

[12] *Placing text boxes — Matplotlib 3.8.4 documentation*. (n.d.). https://matplotlib.org/stable/gallery/text_labels_and_annotations/placing_text_boxes.html

References for Siamese Modeling

[13] Dutt, A. (2022, August 1). Siamese Networks Introduction and Implementation - towards Data Science. *Medium*. https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140e3443dee

[14] *Keras accuracy does not change*. (n.d.). Stack Overflow. https://stackoverflow.com/questions/37213388/keras-accuracy-does-not-change

[15] Rosebrock, A. (2021, April 17). *Building image pairs for siamese networks with Python - PyImageSearch*. PyImageSearch. https://pyimagesearch.com/2020/11/23/building-image-pairs-for-siamese-networks-with-python/

[16] Rosebrock, A. (2024, May 8). *Siamese network with Keras, TensorFlow, and Deep Learning - PyImageSearch*. PyImageSearch. https://pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

[17] *Sklearn.utils.class_weight.compute_class_weight*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html

References for Edge Detection CNN implementation

[18] *Image Kernels explained visually*. (n.d.). Explained Visually. https://setosa.io/ev/image-kernels/

[19] Team, L., & Team, L. (2024, February 5). *Edge Detection using OpenCV | LearnOpenCV #*. LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Code, & Tutorials. https://learnopencv.com/edge-detection-using-opencv/

References for SVM and KNN implementations

[20] Martin, E. (2024, March 18). *Multiclass Classification Using Support Vector Machines*. Baeldung. https://www.baeldung.com/cs/svm-multiclass-classification#:~:text=Multiclass%20Classi

fication-,In%20this%20type%2C%20the%20machine%20should%20classify%20an%20instance%20as,dog%20breed%20in%20an%20image

[21] *Sklearn.svm.SVC*. scikit learn. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[22] *Sklearn.neighbors.KNeighborsClassifier.* scikit learn. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

References for Non-Original Images:

[23] Singh, P. (2021, December 11). Siamese Network Keras for Image and Text similarity. *Medium*. https://medium.com/@prabhnoor0212/siamese-network-keras-31a3a8f37d04

[24] *University of Maryland engineering seal - Bing*. (n.d.). Bing. https://www.bing.com/images/search?view=detailV2&

Appendices

- GANTT Chart: Full GANTT Chart
- Project Development Diagram: Final Data Pipeline
- Weekly Project Progress Log: Weekly Project Log
- README.txt: README.txt