Flooding Damage Detection from Post-Hurricane Satellite Imagery Based on Convolutional Neural Networks Deep Learning for Image and Video Processing

Wasti Murad

February 27, 2024

Contents

Abstract

In this project, I used convolutional neural networks(CNN) to identify flooding damage in the Hurricane Harvey satellite pictures from 2017. I used a Kaggle dataset[1] of images with the labels "damage" and "no damage" for training. I developed and assessed a variety of neural network structures, from simple ones to those that used transfer learning using weights that had already been trained. The ultimate model[2] achieved an accuracy of 0.9775 on test data. I used special tools like 'Superpixel' and 'Lime' functions to make the model show its work in a simple way. Local, state, and federal natural disaster responders might find the method beneficial as it allows

[1]The Kaggle dataset is available at: https://www.kaggle.com/kmader/satellite-images-of-hurricane-damage

[2]Github: https://github.com/muradohi/Flooding-Damage-Detection.git

them to swiftly produce damage assessment maps following a hurricane.

1 Introduction

The frequency of natural disasters has been on the rise since 1980, accompanied by a growing population in disaster-prone regions. As a result, the damages and losses caused by these events have increased significantly(Pi et al. 2020 [1]).

Hurricanes, as tropical storms, pose significant threats to both human lives and property. The swift evaluation of hurricane-induced damage is crucial for first responders. A prompt and effective response is essential to minimize harm and support long-term recovery post a natural disaster. Assessing damage shortly after an event helps in strategically deploying immediate response resources and identifying spatial damage patterns. Moreover, these assessments aid in calculating economic impact estimates, a vital step in securing approval for recovery funding.

However, swiftly and accurately assessing damage right after a disaster comes with its own set of challenges. Access to affected areas may be cut off due to damage to transportation networks. A severe storm can impact a large number of structures across extensive areas, making manual assessment a resource-intensive task, especially when resources are focused on immediate rescue efforts. Having a model that can automatically assess damage using remote sensing imagery captured quickly by a few planes flying over a disaster site would significantly reduce the workload and speed up the delivery of crucial data (location and extent of damaged structures) to responders.

In the past, various machine learning (ML) methods have been employed to address hurricane damage detection, tracking, and estimation. An instance includes the use of different ML algorithms to identify the damage caused by fallen trees on roads during hurricanes. A good accuracy of 86% was achieved for the Hurricane Michael in the Florida region.But this method was not applicable to other hurricanes in the other regions(Gazzea et al. 2021 [2]).

In the modern era of increased computing capacity, deep learning (DL) has witnessed remarkable progress. Specifically, Convolutional Neural Networks (CNN) within the realm of DL have demonstrated exceptional performance in image classification. The CNN model excels at extracting features directly from images, eliminating the need for additional feature extraction methods.

In this project, I explored varied state-of-the-art convolutional neural networks and use them to train and classify the 2017 post-Harvey Hurricane satellite images. I brought an introductory understanding of computer vision by a self-driven exploration into the field. This project served as a compelling opportunity to dive into hands-on practical skills in CNN implementation and experimental setup.

2 Related Works

The author explored the use of CNN models to assess natural disaster damage from aerial imagery. Transfer learning with Volan 2018 videos resulted in eight models achieving 80.69% mAP and 74.48% for high and low altitudes (Pi et al. 2020). A benchmark dataset for Hurricane Harvey in Greater Houston was created for training and testing object detection models (Chen et al. 2018).

Satellite images, deep learning, and post-processing neural networks were combined for disaster response and environmental monitoring, yielding 0.83 accuracy and 0.797 F1 score on the IARPA fMoW dataset (Pritt and Chern 2017). CNNs were employed for change detection in Hurricane Harvey-affected areas, achieving an 81.2% F1 score (Doshi et al. 2018).

Deep Phurie, a CNN model, estimated hurricane intensity with an 8.82 knots RMSE, outperforming the Phurie model (Dawood and Asif 2019). U-Net and ResNet determined road damage in Hurricane

Harvey satellite images with 0.845 accuracy and 0.675 F1 score (Dotel et al. 2020b). VGG-16 CNN/Multilayer perceptron classified time and urgency in Hurricane Harvey 2017 images, showing a 4% accuracy improvement with various CNN model resolutions (Robertson et al. 2019).

3 Dataset

Dataset [3] used in this paper comprises the satellite images obtained after Hurricane Harvey that occurred in the Greater Houston region. The satellite images have been labeled as "damaged" and "undamaged" for the assessment of damage caused to buildings due to the hurricane. The images labelled as "damaged" indicate buildings affected by the hurricane and "undamaged" label indicate the buildings which were unaffected by the hurricane disaster. The number of "damaged "class images are 15,000 and "undamaged" class images are 8000 in number.

Images in the dataset are organized into training, validation, and test sets. The training dataset includes 10,000 images (5,000 no damage, 5,000 damage) , the validation and test sets each include 2,000 images (1,000 no damage, 1,000 damage) and the imbalanced dataset includes 9000 images (1,000 no damage, 8,000 damage) . The filename for each image includes the latitude and longitude coordinates which can be used to plot the locations of the structures in relation to each other.

The data set is very organized and did not need much cleaning or wrangling. Prior to EDA, I did scan all images to verify that none were corrupted. Due to the relatively small number of images in the training dataset, image augmentation was included in the model pipelines .

4 Exploratory Data Analysis

Before delving into neural network training, I took a step back to thoroughly explore the dataset. This exploration was structured into three distinct parts.

4.1 Dataset Creation

Firstly, I examined the folders, noting the quantity of images in each, also the size of the folders. Following this, I constructed a dataset, categorizing the columns into paths, which represented the image paths segregated into train another, validation another, test, and test another folder. To enrich the geographical context, I included columns for latitude and longitude, enabling a visual representation of the images' distribution on a map.

4.2 Visual Inspection

I sampled images from each file to provide a visual glimpse into the dataset's characteristics. This preliminary exploration aimed to lay a solid foundation for understanding the data before moving forward with the neural network training.

Figure 1: Example images from the dataset

4.3 Geographical Distribution

Finally, I mapped out where the images were located geographically to get a sense of how structures are distributed in the dataset. This exploration was key to making informed decisions when it came to training the neural network.

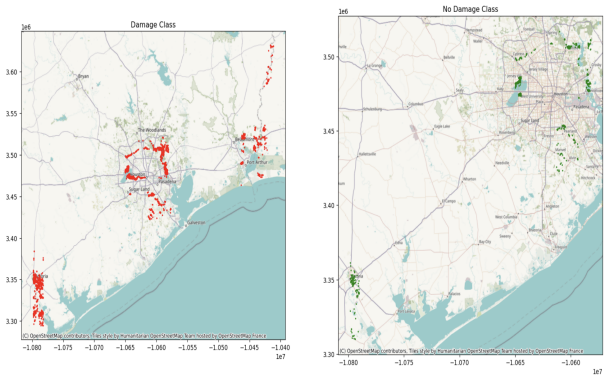


Figure 2: Point locations of Training Images, Damaged or Not

From the figure 2 , it's evident that the training data originates from three distinct regions. In one area, all structures are labeled as damaged, while the other two areas have a mix of damaged and undamaged structures. Within these areas, there are clear spatial patterns indicating where flooding or damage occurred and where it didn't. This observation aligns with the expectation that the impact of the hurricane varied based on factors like topography and urban layout.

However, it's noteworthy that examples from each class often come from entirely different areas. This poses a challenge because when training neural network models, there's a risk that they might learn to identify differences based on the appearance of structures in those specific areas rather than differences attributed to hurricane damage. This insight prompts a careful consideration of potential biases in the dataset during the training process.

5 Model Considerations

One of the most difficult challenges of machine learning is improving its ability to generalize to external data that it has never seen before . Specifically, one significant problem is overfitting the training dataset. Overfitting results when the trained model becomes more flexible than required, leading to unnecessary levels of complexity which results in a worse performance . While the model may seem to perform better on the training set, in reality it becomes too adapted towards the training set and stops generalizing towards previously unseen data. I implemented multiple methods in order to reduce this, such as data augmentation, early stopping, batch normalization, dropout regularization, and weight decay.

5.1 Image Augmentation

Data augmentation is a method to increase the size of data(Perez and Wang 2017[4]) while also improving the quality of these datasets. It has been discovered that larger datasets result in better models , which is one advantage of data augmentation.

In implementing image augmentation, I included rescaling to normalize pixel values within the range [0, 1]. I also applied random rotations, allowing for variations up to 0.1 degrees, along with zooming by 0.1 times and horizontal flipping. These augmentations are designed to introduce diversity into the training data, aiding me in learning robust features and enhancing the model's ability to generalize to unseen images.

5.2 Early Stopping

Early stopping is a technique that stops training a model when it consistently doesn't perform well for a certain number of rounds. In simpler terms, it keeps an eye on the model's performance on a separate validation set and stops training if it notices the model is starting to overfit[5] the training data too much, rather than learning to handle new, unseen data. I used this early stopping approach for all our models, and I set it to wait for 10 rounds without seeing improvement on the validation set before deciding to stop the training. This helps ensure our models don't over-specialize and can better handle diverse data.

5.3 Batch Normalization

Batch normalization is a method which fixes each layer's input's mean and variance [6]. It does this by normalizing each batch, subtracting the batch's mean and dividing by the batch's standard deviation. Ultimately, this decreases training time, allows to use higher learning rates without divergence, and also regularizes the model. I used it to enhance the overall performance of the models.

5.4 Dropout Regularization

Dropout is like adding a bit of randomness to the learning process of a neural network. It works by randomly leaving out certain neurons and their connections while the network is learning. The goal is to avoid the network relying too much on specific neurons, making it more adaptable to different situations.

5.5 Weight Decay

Another method to increase the model's generalization pertains to weight decay, which prevents the parameter weights for each neuron from growing too large unless it is absolutely necessary. It does this by penalizing the cost function for excessively large weights. In our model, I utilized the L2 regularization [7], which adds a penalty term to the cost function: lambda times the sum of the weights squared. Lambda is a hyperparameter that controls how firmly we want to penalize large weights.

6 Modeling

This part explains how I went about training and refining neural network models. I used TensorFlow with the Keras API for developing all models. Both data exploration and model training took place on my personal laptop, providing a cohesive environment for the entire workflow.

To measure how well the models were doing, I checked their accuracy on the validation set, which wasn't used during training. This made sense because there's an equal number of instances for each class. I trained each model for 50 epochs, implementing an early stopping rule that suspended training if no improvement was observed after 10 epochs. The model saved was the one from the epoch displaying the highest validation accuracy.

Considering this is an image classification challenge, I leaned towards using a convolutional neural network. I started with a simple model having three convolution layers and three dense layers. Following this, I fine-tuned some parameters to streamline the model, reducing complexity and training time while preserving performance and promoting convergence.

6.1 Baseline Models

I initially established a straightforward baseline model, comprising three convolution layers, three dense layers, and an output layer with 2 nodes (representing the two classes). The CNN model involved rescaling the input images, resulting in a size of 128*128*3 for the input image. This input was then processed through three convolutional layers with 32 filters each, producing images of size 64*64*32 at each layer.

I implemented the ReLU activation function for all layers except the final output layer. This choice was driven by its computational efficiency and proven effectiveness in various scenarios. Notably, batch normalization was incorporated after the convolution layers but before activation, following experimentation. It was observed that placing batch normalization after activation resulted in a significant drop in validation accuracy.

For updating network weights, the Adam optimizer was employed due to its recognized performance in image classification problems. Adam's adaptive learning rate adjustment over time contributed to its suitability for the task at hand.

The initial model, trained on unaugmented data, achieved a validation accuracy of 0.91700. However, with the integration of image augmentation into the pipeline, there was a notable improvement, and

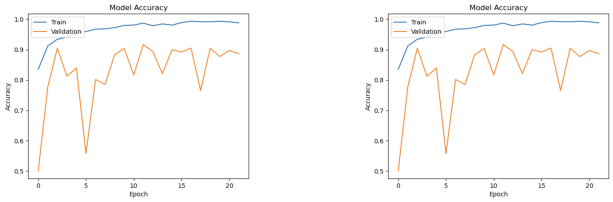


Figure 3: Accuracy and Loss on Validation Data(50 Epochs)

the validation accuracy increased to 0.94300. The remarkable performance boost, particularly for a relatively simple model, suggests distinct patterns between the damage and no damage classes that the network easily picked up.

Despite the positive impact of image augmentation, it's noteworthy that the model's accuracy and validation loss exhibit frequent spikes which can be seen in figure 3, indicating room for improvement. Addressing these fluctuations becomes crucial for refining the model further and ensuring more consistent performance.

6.2 Model Enhancement and Fine-Tuning

Validation accuracy significantly trailed behind training, even though training validation scores typically above 0.98 for the baseline model. Even after adding a variety of training data through image augmentation, this mismatch suggested possible overfitting.To overcome overfitting, I attempted two different approaches to address overfitting in convolutional neural networks: 1) adding max pooling or average pooling layers after each convolution layer, and 2) inserting dropout layers after each dense layer. These techniques not only help mitigate overfitting but also contribute to improved training efficiency.

Pooling layers, like max pooling and average pooling, follow convolutional layers in a neural network. They shrink the width and height of an image, simplifying processing and saving memory. Pooling focuses on the most important information, reducing parameters and preventing overfitting.

Dropout layers defend against overfitting by randomly ignoring nodes in dense layers during training. This introduces randomness, helping the model generalize better. However, it makes the network thinner, so you might need more nodes in dense layers.

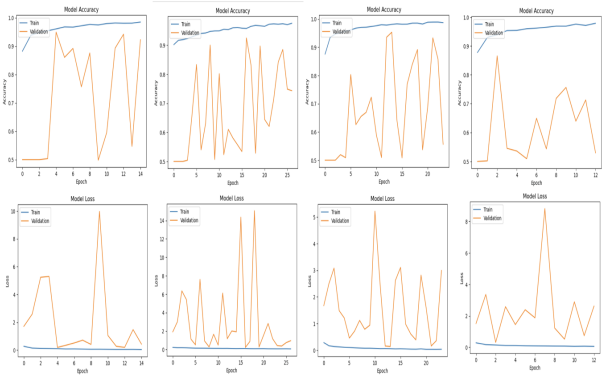


Figure 4: Models Accuracy and Loss on Validation dataset

I tried different combinations of hyperparameters, playing with the size of convolution filters and adjusting the number of nodes in the dense layers. While many models showed good performance, a common issue emerged: models often achieved high accuracy quickly in training but struggled to converge. The figure above shows the training and validation accuracy and loss by epoch for the improved models illustrating this pattern.

Among the enhanced models, the one exhibiting the highest accuracy is the model with fewer dense nodes, reaching an accuracy of 0.95350. However, it's important to note that this model still experiences significant spikes, indicating areas for improvement in stability and consistency.

6.3 Transfer Learning Leveraging Deep Learning Models

After the initial training and evaluation of the models mentioned earlier, I sought to enhance their performance through transfer learning using pre-trained deep learning models. Recognizing the impact of model architecture on results, I decided to explore the capabilities of ResNet-50 [8] and VGG16 [9].

- The choice of ResNet-50 was motivated by its renowned success in addressing the challenges of training very deep neural networks. ResNet-50 introduces residual connections, allowing for the training of deeper networks while mitigating the vanishing gradient problem. This architecture is particularly beneficial when dealing with complex features in images. The utilization of pre-trained weights from a large dataset provided the model with a solid foundation of learned features, potentially leading to improved generalization on our specific task.
- In parallel, I considered the VGG16 architecture, known for its simplicity and effectiveness. VGG16 is characterized by its uniform architecture with small, 3x3 convolutional filters. Despite having fewer parameters compared to ResNet-50, VGG16 has shown strong performance in image classification tasks. The use of transfer learning with VGG16 involved adapting the model's final layers to the specific characteristics of our dataset.

Adding the ResNet50 model, pretrained on ImageNet, was a key step in improving the top-performing model from the previous experiments. I kept the ResNet model weights fixed, only training the additional convolution and dense layers using the structure damage training dataset. Later, I switched to the VGG16 model while maintaining the same settings.

I started with the hyperparameter settings of the best model from the previous section and tried different combinations of convolution filter sizes, dense layer node density, dropout rates, and learning rates through multiple iterations for further refinement. The primary drawback was the extended training duration on the training data, coupled with a substantial runtime when applied to the validation dataset.

6.4 Model Performance Comparison

Table 1 provides a condensed overview of a subset of the assessed iterations, with the exclusion of several for conciseness. It's important to note that all models, excluding the initial baseline, incorporated image augmentation techniques such as rotation, flipping, and zooming within their processing pipelines.

Model	Validation Accuracy
Baseline (no image augmentation)	0.91700
Baseline (with image augmentation)	0.94300
Including Max Pooling (kernel=5) and Dropout Layers	0.94950
Including Avg Pooling (kernel=5) and Dropout Layers	0.92500
Less dense nodes	0.95350
Larger convolution kernel(kernel=10) size	0.86550
TL Model(RESNET50), kernel=3, Strides=1	0.95500
TL Model(VGG16), kernel=5, Strides=2	0.95600

Table 1: Model Performance Comparison

7 Test Model Construction

7.1 Model Architecture

After establishing the previous models, I noticed that models consistently performed well when using a kernel size of 3 and strides of 1. Building on this insight, I created six different models, tweaking various hyperparameters such as learning rate, learning

decay, and activation functions. I also experimented with combining max and average pooling layers. The summarized table provides a quick look at how these adjustments impacted the models' performance.

Model	Validation Accuracy
Non_TL-1: With Decreased Learning Rate	0.97300
Non_TL-2: Small Learning Rate Decay	0.97350
Non_TL-3: Sigmoid Function, Increased Neurons, and Learning Decay	0.96900
Non_TL-4: Increased Neurons, Max and Average Pooling	0.96300
Non_TL-5: No Learning Decay	0.97300
Non_TL-6: Increased Strides	0.96600

Table 2: Test Models Performance Comparison

Initially, when I reduced the learning rate, the model showed remarkable performance with a validation accuracy of 0.973. However, there were noticeable spikes in accuracy and loss. Trying a small learning decay didn't change the accuracy or loss significantly. Experimenting with the Sigmoid function and increasing neurons led to a slight decrease in accuracy, but the upside was fewer spikes. I achieved a similar result by combining increased neurons, max, and average pooling (accuracy of 0.963) with improved stability.

Maintaining the configuration of increased neurons and max and average pooling, I removed learning decay, resulting in an accuracy of 0.973 with minimal spikes. However, when I increased the stride, the model's performance decreased again.

7.2 Performance on Validation Data

7.2.1 Confusion Matrix

To assess and contrast the effectiveness of six distinct models on the validation dataset, I constructed individual confusion matrices. These matrices provide a detailed breakdown of the classification outcomes, revealing the number of accurately identified instances of damaged and undamaged images, as well as the instances where misclassifications occurred.

The objective of this thorough evaluation process is

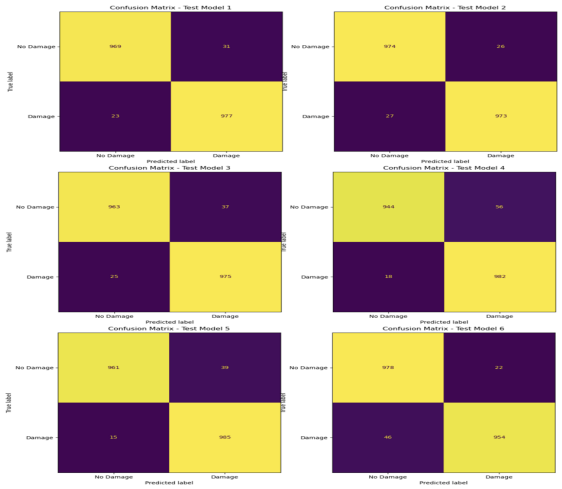


Figure 5: Individual Confusion Matrices

to provide a detailed understanding of the advantages and disadvantages of each model in terms of differentiating between damaged and undamaged images in the validation dataset. Figure 5 provides valuable insights into the performance of all the test models on previously unseen images from the validation set. The results indicate that the model 5 performed admirably, effectively distinguishing between damaged and undamaged instances.

7.2.2 ROC Cuurve

After that, I created a single ROC curve for all test models. This visualization makes it easier to understand how well each model performs in making the trade-off between correctly identifying positive and

negative instances.

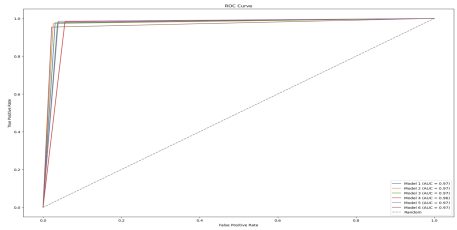


Figure 6: ROC Curve

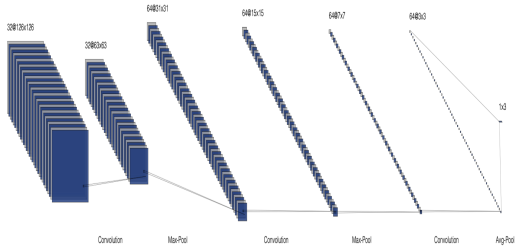


Figure 7: LeNet[10] Style Diagram of the Custom Network

8 Final Model

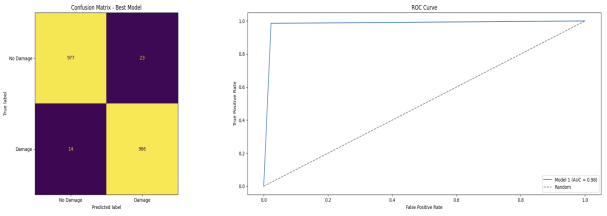
The ultimate model is constructed using a Convolutional Neural Network (CNN) architecture tailored for effective learning. The model initiates with an input layer, followed by convolutional layers with kernel size 3 and varying strides for feature extraction. Max and average pooling layers are strategically incorporated to downsample spatial dimensions and capture hierarchical features. Batch normalization is applied after each convolutional layer to enhance training stability, and rectified linear unit (ReLU) activation functions introduce non-linearity.

The model comprises three convolutional blocks with increasing filter sizes (32, 64, and 64), each followed by the appropriate pooling and normalization operations. The fully connected (dense) layers, with sizes 4096, 1024, and 64, employ ReLU activation functions and dropout regularization (0.3, 0.2, and 0.1, respectively) to prevent overfitting. Dropout randomly excludes neurons during training, promoting better generalization to new data.

The output layer consists of two neurons using the sigmoid activation function, suitable for binary classification tasks. The Adam optimizer is integrated with a learning rate of 0.0001, β_1 of 0.9, β_2 of 0.999, no decay, and not using the AMSGrad variant. These optimizer parameters facilitate efficient weight updates during training.

8.1 Performance on Test Data

The performance of the final model was evaluated using a separate set of test data, comprising an additional 2,000 images kept distinct from the training and validation sets. Remarkably, the final model achieved a test accuracy of 98.15%, surpassing the validation accuracy of 97.30%. The consistent performance on both the validation and test sets suggests the model's robust generalization to previously unseen data.



The confusion matrix for the model's predictions on the test data revealed that false positives were almost twice as common as false negatives.

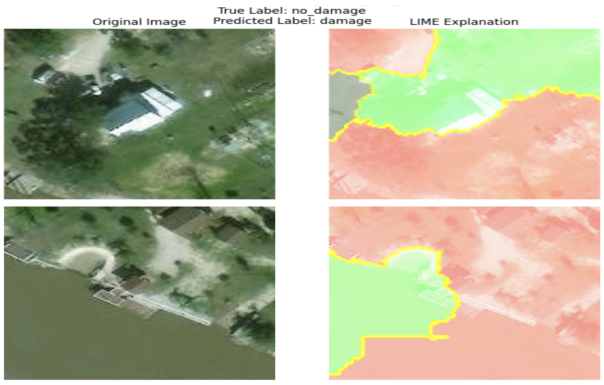
8.2 Explainability

Local Interpretable Model-agnostic Explanations (LIME[11]) is one of the most popular Explainable

AI[12] (XAI) methods used for explaining the working of machine learning and deep learning models. LIME can provide model-agnostic local explanations for solving both regression and classification problems and it can be applied with both structured datasets and even with unstructured datasets like text and images.

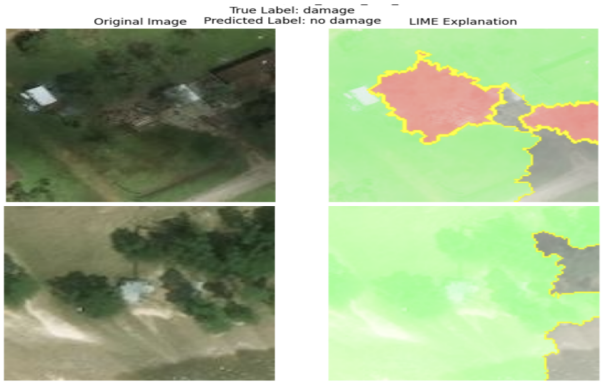
False Positives(23 total, 10 shown):

Using explainability, I dug into why non-damage images were mistakenly labeled as damage. It turns out, the model got confused by surfaces resembling floodwaters, clutter around images, and some low-quality pictures. Notably, false positives were more common in rural structure settings. Examples include images misinterpreted due to their similarity to floodwaters or clutter.



False Negatives (14 total, 10 shown):

For images classified as false negatives, they predominantly feature large or non-residential structures, exhibit considerable variation in the ground surface, and lack apparent signs of floodwater.



9 Final Model (Imbalanced Data):

The final model's performance was assessed using a distinct set of imbalanced test data, consisting of an additional 9,000 images. Among these, 8,000 were labeled as "damage," and 1,000 were labeled as "no damage," carefully partitioned from the training and validation sets. Remarkably, the model attained an

accuracy of 97.53%, closely mirroring the accuracy achieved on a balanced test dataset. Given the class imbalance in our dataset, additional metrics such as F1 score, precision, and recall were employed alongside accuracy for a comprehensive evaluation.

Performance Metrics	Test Model 5
Accuracy	0.975333
Precision	0.99554
Recall	0.976625
True Positive Rate	0.976625
False Positive Rate	0.035

Table 3: Performance Metrics Report

10 Future Work

The outcome of this work was a model that could distinguish between aerial photos of damaged and undamaged structures with a high degree of accuracy. The model would be most helpful if it were a component of a pipeline that could automatically input much bigger aerial photos, crop those images to just the specific structures, and then classify them. This could be achieved rather quickly, given building detection methods are already fairly precise, either by utilizing an existing building footprint dataset or by adding a building detection algorithm at the beginning of the process.

I also wish to expand the current research to road damage annotation which could help plan effective transportation routes of food, medical aid, or energy to the disaster victims.

11 Conclusion

In this paper, a thorough literature review and implementation of - 'Damage Detection of buildings and roads using Satellite Images' has been carried out. Many existing research works show errors while detection due to an over- fitting problem, thereby misclassifying damaged structures as not damaged techniques. This paper tries to minimize the overfitting issues using different techniques and hyper parameter optimisation. This effort produced a model that performs well on both balanced and imbalaned test data. However, there are some important caveats and limitations to be aware of, which could be addressed through access to improved (or different) training data. Many (but not all) of the images depicting damaged structures included visible flood water, suggesting that the model likely learned to identify flood water (among other features). A model that could identify structure damage even after flood waters have subsided would be even more useful but would need to identify much more subtle attributes in the images. This could be accomplished with labeled training images from imagery several weeks after a hurricane, rather than directly after.

In working on this project, I got the hang of tweaking different settings in my CNN model to make it work better. It was great learning why a model might get too good at the training pictures (overfitting) and how using dropout layers, early stopping can fix that. Plus, I used Lime and Superpixel [13] tools, which were new to me, to show how the model looks at different parts of the pictures when making predictions. It's like giving the model a translator for its thoughts. This whole experience not only made me better at handling CNNs but also helped me explain them in a way that makes more sense.

Bibliography

[1] Yalong Pi, Nipun D Nath, and Amir H Behzadan. Convolutional neural networks for object detection in aerial imagery for disaster response and recovery. *Advanced Engineering Informatics*, 43:101009, 2020.

[2] Michele Gazzea, Alican Karaer, Mahyar Ghorbanzadeh, Nozhan Balafkan, Tarek Abichou, Eren Erman Ozguven, and Reza Arghandeh. Automated satellite-based assessment of hurricane impacts on roadways. *IEEE Transactions on Industrial Informatics*, 18(3):2110–2119, 2021.

[3] Quoc Dung Cao and Youngjun Choe. Detecting damaged buildings on post-hurricane satellite imagery based on customized convolutional neural networks, 2018. URL https://dx.doi.org/10.21227/sdad-1e56.

[4] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[5] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[7] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*, 2012.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[10] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019. doi: 10.21105/joss.00747. URL https://doi.org/10.21105/joss.00747.

[11] Pavan Rajkumar Magesh, Richard Delwin Myloth, and Rijo Jackson Tom. An explainable machine learning model for early detection of parkinson's disease using lime on datscan imagery. *Computers in Biology and Medicine*, 126: 104041, 2020.

[12] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. Explainable ai: the new 42? In *Machine Learning and Knowledge Extraction: Second IFIP TC 5, TC 8/WG 8.4, 8.9, TC 12/WG 12.9 International Cross-Domain Conference, CD-MAKE 2018, Hamburg, Germany, August 27–30, 2018, Proceedings 2*, pages 295–303. Springer, 2018.

[13] Ludwig Schallner, Johannes Rabold, Oliver Scholz, and Ute Schmid. Effect of superpixel aggregation on explanations in lime–a case study with biological data. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 147–158. Springer, 2020.