

Damage Assessment in the Aftermath of Natural Disasters: A Deep Learning Approach

Ellemieke Van Kints
University of Georgia
Athens, Georgia
ejv88036@uga.edu

Abstract—The objective of this research is to build a deep learning model for classifying satellite images based on their structural damage. Studies have shown that changes in climate conditions, especially the warming of global temperatures, are increasing the intensity of weather-related natural disasters [1]. Hurricane Ian, in particular, caused more casualties than any other hurricane in Florida since 1935 [2], and many communities waited days to receive aid due to the sheer destruction of the storm. Therefore, in order to support natural disaster response efforts, I developed a deep learning model that automatically identifies whether an image contains structural damage or not. This would eliminate the need for field surveys of disaster conditions, as damage assessment can be remotely with satellite imagery.

Data was collected from the xView2 building damage assessment (xBD) dataset, which contains high-resolution satellite imagery from 10 large-scale natural disaster events around the world [3]. My proposed method performs binary classification, assigning images to damage or no-damage categories. The final model is composed of a convolutional neural network with three convolution layers, three dense layers and an output layer with two nodes, corresponding to the two classes. To improve model performance and reduce overfitting, max pooling layers were added after each convolution layer and dropout layers after each dense layer. The final model achieved an accuracy of 0.9266 on test data, thereby performing fairly accurate damage assessment. In order to demonstrate the real-world applications of this research, the model was also tested on satellite imagery from Fort Myers captured before and after Hurricane Ian, also achieving an accuracy of 0.9266. Based on these results, this method could be used in disaster planning to quickly develop damage assessment maps and help first responders prioritize affected locations. However, there is substantial room for improvement in this research, and future work can be done to increase model accuracy.

I. INTRODUCTION

After Hurricane Ian struck Florida in late September of 2022, it left at least 119 dead, more than any other hurricane in Florida had caused since 1935 [2]. Many disaster response teams were overwhelmed and unable to reach civilians due to the sheer scale of the storm. In the aftermath, bridges had collapsed, cellular service providers were down and entire neighborhoods were washed away. Timely and accurate damage assessment is critical to response efforts, especially after storms as destructive as Hurricane Ian. However, disaster recovery researchers agree that “due to the complex and dangerous situations in affected areas, field surveys of disaster

conditions are not always feasible” [4]. Remote sensing overcomes this problem, as it allows for faster damage detection across greater areas and can help response teams efficiently allocate relief assistance.

Therefore, in order to aid in natural disaster response, I propose a deep learning model which automatically identifies whether an image contains structural damage or not. This would allow response teams to assess post-disaster damages in a non-intrusive manner. Training data is collected from various large-scale natural disasters, including hurricanes, floods, earthquakes, tsunamis, wildfires and volcanic eruptions. My proposed method performs binary classification, assigning images to damage or non-damage categories. For some background, know that image classification is a supervised learning problem, wherein a machine learning model is trained using a set of pre-labeled data. The task is to assign an image to a specific class. The classification results from the model are checked against the true class label, and the model updates accordingly. This, in turn, creates a trained model with a set of weights learned by continuously connecting images to their correct class labels.

In this research, the class labels are damage and no-damage, and a convolutional neural network is used to classify the images. Several models are trained and evaluated, with the best model achieving an accuracy of 0.9266 on test data. In order to demonstrate real-world applications, the model was also tested on satellite imagery from Fort Myers captured before and after Hurricane Ian, also achieving an accuracy of 0.9266. However, in most machine learning models, an accuracy > 0.95 is desired. Therefore, my model was only moderately accurate at predicting whether an image contains structural damage or not. This may be due to the model overfitting the relatively small training dataset, but future work can be done to address this problem by using image augmentation to generate more data. Ultimately, my research aims to answer the question: How can we leverage remote sensing imagery to effectively and automatically assess damages in the aftermath of natural disasters?

II. RELATED WORK

Previous research shows that image classification using deep learning has outstanding performance in the field of remote sensing. In the context of damage detection, deep learning

models have regularly outperformed traditional field survey methods. Shao et al. developed a building damage detection network (BDD-Net) composed of a novel end-to-end remote sensing pixel-classification deep convolutional neural network [4]. BDD-Net classifies each pixel in a post-disaster image as either non-damaged building, damaged building, or background. Their results show that BDD-Net is effective for recognizing buildings damaged by different disasters in different areas [4]. My research takes this idea and extends it to whole images, rather than individual pixels, for simplicity during model development.

Zheng et al. proposed a deep object-based semantic change detection pipeline, called ChangeOS, which integrates building localization and damage classification into a unified end-to-end framework [5]. ChangeOS segments the individual buildings in an image and classifies them based on their damage. This approach leverages the pre and post-disaster images for one region. Based on the building localization results from the pre-disaster image, damage classification involves assigning the damage level label to each building instance in the post-disaster image. My work skips the building localization step, and instead, assign the class label to the whole image. This trades off accuracy for speed. However, future work can be done to integrate this step into my overall pipeline to obtain higher accuracy and a more detailed damage assessment.

Kapoor trained and evaluated several neural network architectures to determine the most efficient damage assessment method [6]. His models were trained using satellite images captured in the days after Hurricane Harvey. He found that the basic convolutional neural network, with max pooling layers added after each convolution layer and dropout layers added after each dense layer, performed best. The model achieved an accuracy of 0.9775 on test data, therefore demonstrating accurate post-hurricane damage assessment. However, since the model was trained on data from one specific storm, it is not generalizable to other types of natural disasters in other areas. My research builds upon the work done by Kapoor. I follow his same structure, but my models are trained on satellite images from various natural disasters across the world, making them generalizable to future events.

III. SUMMARY OF DATASET

Image data was obtained from the xView2 building damage assessment (xBD) dataset [3]. The xBD dataset contains pre and post high-resolution satellite imagery from 10 large-scale natural disaster events around the world, recorded between 2016 and 2019. The disaster types in the dataset include hurricane, flood, earthquake, tsunami, wildfire and volcanic eruption. The dataset contains 4,665 pairs of high-resolution RGB satellite imagery in PNG format. Each pair is considered an instance, with one image pre-disaster and its matching image post-disaster. An example of an instance pair can be seen in *Figure 1* below.

The xBD dataset was already split into appropriate training, validation and testing groups. The training dataset contained 2,799 instance pairs, the validation dataset contained 933



Fig. 1. An example of an instance pair in the xBD dataset.

instance pairs and the testing dataset contained 933 instance pairs. Label information (polygon annotations and metadata) and ground truth targets (pixelwise values in PNG format) were also provided, but this data was omitted, as it was not in the scope of this research. During data preprocessing, I reorganized the dataset to better suit the classification model I wished to build. This step is discussed in detail in the *Data Preprocessing* section, where the final dataset is further explored.

The Hurricane Ian data was collected manually from various locations in Fort Myers using Google Earth [13]. Both pre and post-disaster images were gathered. The dataset contains 10 instances, 5 damage images and 5 no-damage images. The dataset is small because it is only being used for testing purposes. All images are in RGB and PNG format, like those in the xBD dataset. The data is organized into damage and no-damage categories, corresponding to the two class labels. The purpose of this dataset is to determine whether my final model is generalizable to recent natural disaster events.

IV. SOFTWARE SYSTEM

A. Software Overview

The goal of my software is to build multiple machine learning models that can accurately classify satellite images into damage and no-damage categories. I train a series of convolutional neural networks and evaluate their performance on test data. This approach is based off the work done by Kapoor in his damage detection models for Hurricane Harvey [6]. I am following the same structure, but my models are trained on satellite images from various natural disasters across the world, making them generalizable to future events. Also, instead of comparing three entirely different models like Kapoor, my models build upon each other, adding an additional method each iteration to improve performance. This way, I am able to illustrate the effectiveness of certain machine learning techniques. The three models I evaluate are briefly described below.

- 1) A model with 3 convolution layers and 3 dense layers
- 2) Model 1, with max pooling layers added
- 3) Model 2, with dropout layers added

After training, I tune the hyperparameters on the best performing model to further increase accuracy. From this, I

learn that decreasing the convolution kernel size and reducing the number of nodes in the dense layers by 50% improves performance. The model achieves much higher accuracy than the baseline model, but it is unfortunately not converging, which suggests lack of generalizability.

Lastly, the model with the highest validation accuracy is evaluated on test data to achieve a final accuracy score. To demonstrate the real-world applications of my research, this model is also tested on data from Fort Myers captured before and after Hurricane Ian.

The programming language Python and frameworks Tensorflow [7], Keras [8], Scikit-learn [9], Pandas [10], NumPy [11] and Pillow [12] were used to create the software system.

B. Data Preprocessing

In order to upload my dataset to Google Drive, its size had to be significantly reduced. Each image in the dataset had a 1,024 x 1,024 resolution, and with 9,330 images total, the file size was far too large to fit on Google Drive. So, the first step I took in my research was reducing the file size of my dataset. I tried various techniques, but found compressing each image to 512 x 512 pixels to be the best solution, as it allowed me to retain all the data.

My other approach was to manually delete images from the dataset. However, I quickly realized that doing so would disrupt the balance between disaster types in each class. The original dataset had a fairly uniform distribution of disaster types, and I was concerned that deleting images would cause one disaster type to influence my model more so than the others. So, I decided the approach was infeasible. Furthermore, I discovered that even if I were to delete half of the dataset, it would still be too large to fit on the Google Drive. I wanted to retain the original dataset as much as possible, because a smaller training set would lead to overfitting. Thus, I realized that simply compressing each image would allow me to significantly reduce the file size without deleting instances from the dataset.

After compressing the dataset, I reorganized it to better suite the classification models I wished to build. The data was already organized into training, validation and testing sets, so all that was left to do was sort the images into *damage* and *no-damage* categories, corresponding to the two class labels. The structure of the dataset can be seen in *Figure 2* below.

```
data/
  train/
    damage/
    no-damage/
  valid/
    damage/
    no-damage/
  test/
    damage/
    no-damage/
```

Fig. 2. The directory structure for the dataset.

Because the dataset was originally organized by instance pairs, with one pre-disaster image and one post-disaster image,

splitting up the data into *damage* and *no-damage* groups automatically resulted in balanced classes. This is illustrated in *Figure 3* below.

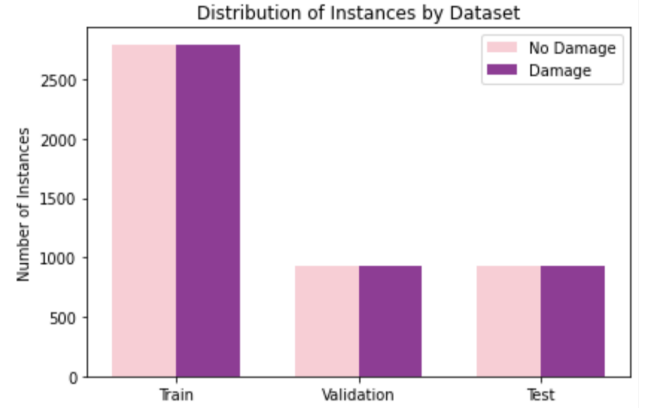


Fig. 3. The distribution of *damage* and *no-damage* instances in each training, validation and testing dataset.

C. Model Training

The main software system trains three separate models. Given the image classification problem, it was assumed that a convolutional neural network architecture would perform best. Starting with a very simple model featuring three convolution layers and three dense layers, I gradually added layers until there was no longer a gain in accuracy. Then, I experimented with altering hyperparameters to further improve performance.

All models were developed with TensorFlow [7] using the Keras API [8]. Model performance was evaluated using accuracy on the validation set, which is an appropriate metric because there are an equal number of instances in each class. Like Kapoor, I trained each model for 50 epochs, with early stopping if the model did not improve after 10 epochs [6]. The weights from the best performing epoch were saved.

I first created a baseline model with three convolution layers, three dense layers and an output layer with 2 nodes, corresponding to the two classes. This is the same model architecture that Kapoor used in his research [6]. I refer to this model as *Model 1*. Details of the model architecture are summarized in *Table 1* below. Like Kapoor, some initial modeling decisions were made here and carry through to the other models:

- The ReLU activation function is used for all layers except the final output layer.
- Batch normalization is included after the convolution layers but before activation.
- Adam optimization is used for updating network weights, because it is known to perform well on image classification problems.

Layer
Rescaling
Convolution (filters=32, kernel_size=5, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Batch Normalization
ReLU Activation
Flattening
Dense (1,024 nodes, ReLU activation)
Dense (512 nodes, ReLU activation)
Dense (256 nodes, ReLU activation)
Dense (2 nodes, ReLU activation)

Table I. The architecture for Model 1, the baseline model.

Model 1 achieved a validation accuracy of 0.66131. This was relatively poor performance, suggesting that the network had a difficult time learning the differences between the damage and no damage classes. The training accuracy for *Model 1* regularly exceeded 0.95, while the validation accuracy remained substantially lower. This suggested that the model was overfitting to the training data. Kapoor shares, "two common ways to reduce overfitting for convolutional neural networks are to 1) add max pooling layers after each convolution layer and 2) add dropout layers after each dense layer" [6]. The following models leverage these techniques to increase model performance.

Max pooling layers down sample the features detected after the convolution layers by applying a filter that sets each pixel to the maximum value within its window. This generalizes the features detected in the image, thus reducing the sensitivity of the model. *Model 2* applies this method by adding max pooling layers after each convolution layer. This model achieved a validation accuracy of 0.84137, which is a substantial improvement from the previous model. Here, we can see how the addition of max pooling layers directly impacts the performance of the model. By generalizing the feature map of the input image, the model became more resilient to overfitting. Details of the model architecture are summarized in *Table 2* below.

Layer
Rescaling
Convolution (filters=32, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Flattening
Dense (1,024 nodes, ReLU activation)
Dense (512 nodes, ReLU activation)
Dense (256 nodes, ReLU activation)
Dense (2 nodes, ReLU activation)

Table II. The architecture for *Model 2*, with max pooling layers added after each convolution layer.

To further improve the model, dropout layers were added after each dense layer. Machine learning experts share that "dropout layers simulate having a large number of different network architectures by randomly dropping out nodes during training. This forces the nodes within a layer to probabilistically take on more responsibility for the input," thus reducing the chances of the model overfitting to the training data [14]. Furthermore, "dropout layers offer a very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error" [14]. In *Model 3*, dropout layers were added after each dense layer. The percentage of nodes dropped after each layer decreases by 0.20 each time, which is similar to what is done in previous work [6]. *Model 3* achieved a validation accuracy of 0.85959. This is a slight improvement from the previous model, but nonetheless, it can be seen how the addition of dropout layers impacts the performance of the model. By simulating a sparse activation in the dense layers, "the network breaks up situations where layers adapt to correct mistakes from prior layers, in turn making the model more robust" [14]. Details of the model architecture are summarized in *Table 3* below.

Layer
Rescaling
Convolution (filters=32, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=5, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Flattening
Dense (1,024 nodes, ReLU activation)
Dropout (rate=0.5)
Dense (512 nodes, ReLU activation)
Dropout (rate=0.3)
Dense (256 nodes, ReLU activation)
Dropout (rate=0.1)
Dense (2 nodes, ReLU activation)

Table III. The architecture for *Model 3*, with dropout layers added after each dense layer, with the exception of the final layer.

D. Model Improvement and Refinement

Several combinations of hyperparameters were tested on *Model 3*, with the goal of further improving performance. I first ran experiments with smaller and larger convolution filters, then I ran experiments with less and more nodes in the dense layers. This follows what Kapoor did during his model refinement [6]. From these experiments, I found that decreasing the convolution kernel size and reducing the number of nodes in the dense layers by 50% substantially improves performance. These models achieved accuracies of 0.86817 and 0.88264, respectively. However, while they performed better than the previous models, a recurring issue was that these models would achieve high accuracy after only a few training epochs and never converge. *Figure 4* summarizes training and validation accuracy by epoch for a model that demonstrates this trend.

This lack of convergence suggests that the models may not be generalizable to the test set and other unseen data. I attempted to solve this problem with adjustments to the model architecture but was ultimately unsuccessful. However, during my experimentation, I was able to surpass the > 0.90 threshold and achieve a 0.9239 validation accuracy. I refer to this model as *Model 4*, and its high accuracy was achieved by adding a dense layer with 1,024 nodes, followed by a dropout layer with a 0.7 dropout rate. I also let *Model 4* train for 100 epochs instead of 50, using the same early stopping rule. But, despite the high accuracy, the model still failed to converge, thus indicating a lack of generalizability. These results are summarized in *Figure 5*.

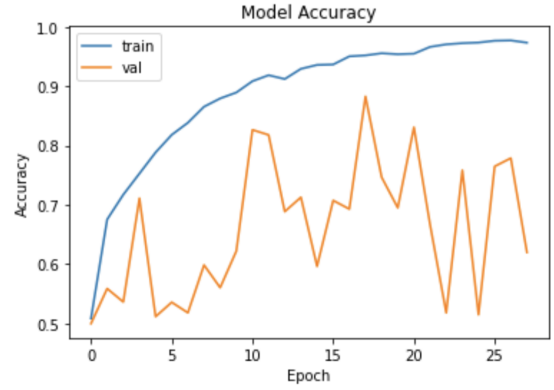


Fig. 4. The training and validation accuracy by epoch for a model I built during hyperparameter tuning. This illustrates the lack of generalizability of the model due to its failure to converge.

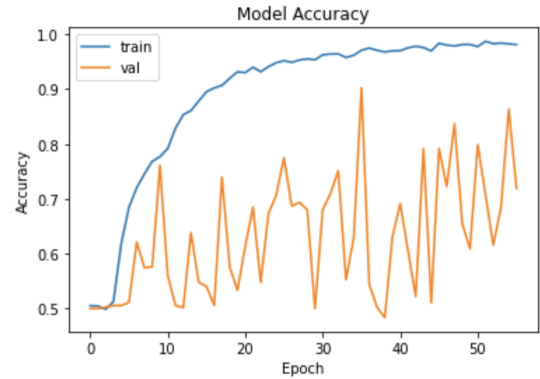


Fig. 5. The training and validation accuracy by epoch for *Model 4*, the final model. This illustrates the lack of generalizability of the model due to its failure to converge.

Details of the model architecture can be seen in *Table 4* below.

Layer
Rescaling
Convolution (filters=32, kernel_size=3, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=3, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Convolution (filters=64, kernel_size=3, strides=2)
Max Pooling (pool size=2, strides=2)
Batch Normalization
ReLU Activation
Flattening
Dense (1,024 nodes, ReLU activation)
Dropout (rate=0.7)
Dense (512 nodes, ReLU activation)
Dropout (rate=0.5)
Dense (256 nodes, ReLU activation)
Dropout (rate=0.3)
Dense (128 nodes, ReLU activation)
Dropout (rate=0.1)
Dense (2 nodes, ReLU activation)

Table IV. The architecture for *Model 4*, with an additional dense and dropout layer.

V. RESULTS

A. Comparing Models

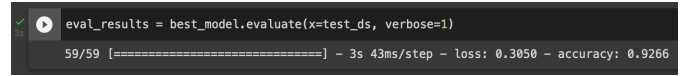
The models I trained produced accuracy scores ranging from 0.66131 for the first model to 0.9239 for the last model. My results show that a standard convolutional neural network, with added max pooling and dropout layers, outperforms the baseline method. The addition of the max pooling layer, in particular, resulted in the biggest jump in accuracy, from 0.66131 in *Model 1* to 0.84137 in *Model 2*. The table below summarizes my results. Because the validation accuracy rates were highest for *Model 4*, it was selected as the final model.

Model	Validation Accuracy
Model 1	0.66131
Model 2	0.84137
Model 3	0.85959
Model 4	0.9239

Table V. The models and their corresponding validation accuracy.

B. Final Model Performance

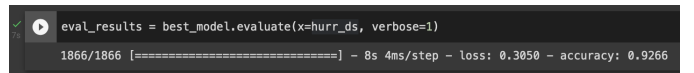
The final model achieved an accuracy of 0.9266 on test data, which was slightly better than the accuracy on validation data. In other words, the model is able to correctly classify the test data into damage and no-damage categories with 0.9266 accuracy. The output from this test can be seen in *Figure 6* below. However, this accuracy score was below my expectations, as my model was unable to reach the > 0.95 threshold. This may be due to the model overfitting to the relatively small training dataset.



```
eval_results = best_model.evaluate(x=test_ds, verbose=1)
59/59 [=====] - 3s 43ms/step - loss: 0.3850 - accuracy: 0.9266
```

Fig. 6. The program output from testing *Model 4* with the test data.

In order to demonstrate the real-world applications of my research, I also tested the model on data from Fort Myers captured before and after Hurricane Ian. The model also happened to achieve an accuracy of 0.9266 on the Hurricane Ian dataset, indicating that it is can perform fairly accurate damage assessment on new data. The output from this test can be seen in *Figure 7* below.



```
eval_results = best_model.evaluate(x=hurr_ds, verbose=1)
1866/1866 [=====] - 8s 4ms/step - loss: 0.3850 - accuracy: 0.9266
```

Fig. 7. The program output from testing *Model 4* with the Hurricane Ian data.

VI. CONCLUSIONS

These models were created to classify satellite images based on the severity of their structural damage. Given that the highest accuracy on test data was 0.9266, these models were fairly accurate in predicting whether an image contains structural damage or not. However, their lack of convergence suggests that they are not necessarily generalizable to new data. This is mostly due to the relatively small training dataset, but future work can be done to improve model convergence by using image augmentation to generate more data. Also, further hyperparameter tuning could reveal better performances, as I experimented with only convolutional kernel size and the number of nodes in the dense layers. Overall, machine learning

models have great potential in image classification problems, but lots of data and experimentation is required to find an accurate method.

REFERENCES

- [1] "Increase in natural disasters on a global scale by ten times," Vision of Humanity, 27-Oct-2020. [Online]. Available: <https://www.visionofhumanity.org/global-number-of-natural-disasters-increases-ten-times/>. [Accessed: 04-Dec-2022].
- [2] M. Smith, F. Robles, E. Fawcett, and A. Sasani, "Many of hurricane Ian's victims were older adults who drowned," The New York times, The New York Times, 07-Oct-2022.
- [3] R. Gupta et al., "XBD: A dataset for assessing building damage from satellite imagery," arXiv [cs.CV], 2019.
- [4] J. Shao, L. Tang, M. Liu, G. Shao, L. Sun, and Q. Qiu, "BDD-Net: A General Protocol for Mapping Buildings Damaged by a Wide Range of Disasters Based on Satellite Imagery," Remote Sensing, vol. 12, no. 10, p. 1670, May 2020, doi: 10.3390/rs12101670.
- [5] Z. Zheng, Y. Zhong, J. Wang, A. Ma, and L. Zhang, "Building damage assessment for rapid disaster response with a deep object-based semantic change detection framework: From natural disasters to man-made disasters," Remote Sens. Environ., vol. 265, no. 112636, p. 112636, 2021.
- [6] A. Kapoor, hurricane_damage: Post-Hurricane Structure Damage Assessment Based on Aerial Imagery Leveraging Convolutional Neural Networks.
- [7] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," arXiv [cs.DC], 2016.
- [8] F. Chollet et al., Keras: Deep Learning for Humans.
- [9] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," arXiv [cs.LG], 2012.
- [10] W. McKinney, "Data Structures for Statistical Computing in Python," in Proceedings of the 9th Python in Science Conference, 2010.
- [11] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, 2020.
- [12] A. Clark, Pillow (PIL Fork) Documentation. readthedocs, 2015. [Online]. Available: <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>
- [13] Google, "Fort Myers." Accessed Dec. 6, 2021 [Online]. Available: https://earth.google.com/web/search/Fort+Myers,+FL/@26.61863379,-81.83229817,5.14901553a,33252.84679951d,35y,0.00004001h,0t,0r/data=CigiJgokCR4dFTxchDpAEWzAAyY6gzpAGdcEkx_bfVTAIZitMEgwflTA
- [14] Machinelearningmastery.com. [Online]. Available: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>. [Accessed: 05-Dec-2022].