VIT-AP UNIVERSITY

Post-Hurricane Building Damage Detection

CSE4006 – DEEP LEARNING

PROJECT REPORT

Class Number – **AP2024254000704**

SLOT – **D1+TD1**

Course Type – **EPJ**

Course Mode – **Project Based Component (Embedded)**

Department of Artificial Intelligence and Machine Learning

School of Computer Science and Engineering

By

22BCE7788 Krishna Nand Jha

Submitted to:-

Prof.Rajeev Sharma

Associate Professor, SCOPE, VIT-AP.

2024 -2025

TABLE OF CONTENTS

ABSTRACT

Natural disasters, particularly hurricanes, often result in large-scale devastation to residential areas and critical infrastructure. In the immediate aftermath of such events, accurately and swiftly assessing the extent of damage is crucial for effective disaster response and recovery planning. Traditionally, this process has relied on manual, ground-based inspections, which—while reliable—are time-consuming, labor-intensive, and frequently hindered by logistical challenges such as debris, flooding, or unsafe conditions.

Recent advancements in remote sensing and artificial intelligence have opened new frontiers in automating post-disaster damage assessment. High-resolution satellite imagery, when combined with the power of deep learning, offers a scalable and efficient solution to evaluate structural damage across vast geographic regions in near real-time.

This project proposes a deep learning-based approach to detect and classify building damage resulting from hurricanes using satellite imagery. Specifically, it aims to develop a convolutional neural network (CNN) model capable of analyzing pre- and post-disaster satellite images to determine whether a building is damaged or undamaged, and, optionally, to further categorize the severity of the damage.

By automating this process, the system supports emergency response teams, humanitarian organizations, and governmental agencies in making informed decisions and prioritizing relief efforts based on real-time damage intelligence. The ultimate goal is to enhance situational awareness, reduce response time, and improve the overall effectiveness of disaster management operations.

Chapter 1
INTRODUCTION

1.1 Objective

The primary objective of this project is to design and implement a deep learning-based system capable of automatically assessing hurricane-induced damage to residential buildings using satellite imagery. The specific goals include:

- Developing convolutional neural network (CNN) models—including a baseline CNN and a regularized variant—for binary classification of building damage (damaged vs. undamaged).

- Utilizing publicly available post-hurricane satellite images to detect visual indicators of structural damage.

- Creating an efficient image preprocessing and classification pipeline suitable for large-scale satellite datasets.

- Evaluating model performance using key classification metrics such as accuracy, precision, recall, and F1-score.

- Exploring the impact of class imbalance on damage detection performance by comparing results on both balanced and unbalanced datasets.

This system is aimed at supporting rapid post-disaster analysis, reducing the reliance on manual damage surveys, and accelerating humanitarian response and recovery operations.

1.2 Scope & Motivation

1.2.1 Scope

This project focuses specifically on post-hurricane structural damage classification in residential areas using post-event satellite imagery. The project scope includes:

- Working with publicly available, annotated satellite datasets.
- Implementing and evaluating deep learning models (CNN-based) for binary classification.
- Addressing class imbalance and testing model generalization across different dataset conditions.
- Using visualizations and evaluation metrics to interpret model performance.

Areas outside the scope:

- Real-time or multi-temporal (pre- and post-event) satellite data acquisition.
- Damage detection for non-residential infrastructure (e.g., roads, utilities).
- Assessment of disasters other than hurricanes (e.g., floods, wildfires, earthquakes).

1.2.2 Motivation

With the increasing frequency and severity of hurricanes, there is a growing need for efficient, scalable, and automated tools to assess disaster impact. Traditional damage assessments—conducted via on-ground surveys or aerial inspections—are often slow, costly, and pose risks to personnel.

Given the growing availability of high-resolution satellite imagery and the success of deep learning in image classification, this project leverages these advancements to automate post-disaster damage detection. The end goal is to assist emergency responders and planners with actionable insights that improve the speed and coordination of disaster response.

1.3 Organization of the Report

This report is structured to provide a comprehensive overview of the project, its

methodologies, results, and potential applications. The following sections are included:

• Chapter 1: Introduction – This chapter outlines the objective, scope, and motivation behind the project, providing the necessary background and context for the work undertaken.

• Chapter 2: Literature Review – A review of the existing research on post hurricane building damage detection, and the applications of deep learning in image classification. This section highlights previous work and identifies the gaps that this project aims to fill.

• Chapter 3: Methodology – In this chapter, the technical approach taken to build the damage detection system is discussed. This includes details on the dataset used, preprocessing steps, model architecture (CNN), and training methods.

• Chapter 4: Model Training and Evaluation – This section covers the training process of the deep learning model, including the evaluation metrics used to assess its performance. It also discusses the results obtained during the testing phase.

• Chapter 5: Results and Discussion – This chapter presents the results of the experiments and provides an analysis of the findings. It also discusses the strengths and weaknesses of the model and how it can be improved.

• Chapter 6: Future Work and Applications – This section outlines potential future improvements to the project, such as expanding the dataset, improving model accuracy.

• Chapter 7: Conclusion – A summary of the work done, the outcomes of the project, and the potential impact of the usage for future hurricane incidents.

• References – A list of all the scholarly articles, papers, books, and other resources referenced throughout the report.

• Appendices – Additional information such as source code, dataset details, and extra figures or charts that support the project.

Chapter 2
LITERATURE SURVEY

2.1 Traditional Damage Assessment Approaches

Accurate damage assessment after hurricanes is essential for effective disaster response. The integration of remote sensing and machine learning, especially deep learning, has revolutionized how quickly and accurately we can perform this analysis.

Traditional methods involve manual inspections and aerial surveys, which, while accurate, are resource-intensive and slow. Remote sensing introduced scalable alternatives using satellite imagery and basic image analysis techniques such as pixel-level comparison and change detection. However, these methods depend heavily on manual feature selection and do not generalize well across regions or disaster types.

2.2 Deep Learning in Remote Sensing

Deep learning, particularly CNNs, has emerged as a powerful approach for analyzing satellite images. CNNs automatically extract spatial and structural patterns from raw images, making them ideal for detecting damage in complex urban environments. They have been widely applied in various remote sensing applications including object detection and classification.

In post-disaster contexts, CNNs have been used to classify damage severity and identify damaged structures using annotated satellite datasets. The effectiveness of CNNs in such tasks has been demonstrated across multiple studies due to their ability to model spatial hierarchies and textures.

2.3 Existing Research and Applications

Key works in this field have demonstrated various deep learning strategies for disaster response:

- **Gupta et al. (2019):** Used CNNs with fused RGB and SAR satellite data to improve damage classification.
- **Doshi et al. (2020):** Developed lightweight CNN architectures optimized for disaster detection on the xView2 dataset.
- **Xu et al. (2021):** Proposed dual-stream CNNs using pre- and post-event imagery to enhance structural change detection.

These studies underscore the potential of CNNs in building automated systems that can deliver timely and reliable damage assessments.

2.4 Challenges and Research Gaps

While promising, several challenges persist:

- **Limited Labeled Data:** High-quality labeled satellite datasets for disasters remain scarce and expensive to curate.
- **Class Imbalance:** Damaged buildings are often underrepresented in datasets, leading to biased models.
- **Generalization Issues:** Models trained on a specific region or disaster type may not perform well in other contexts.
- **Image Quality Issues:** Satellite images may suffer from occlusions (e.g., clouds, shadows), impacting prediction accuracy.

This project directly addresses class imbalance and limited-data scenarios by experimenting with balanced and unbalanced datasets and using regularization techniques to improve model generalization.

Chapter 3
HARDWARE AND SOFTWARE REQUIREMENTS

3.1 Hardware Requirements

To implement and evaluate the CNN-based damage detection models efficiently, the following hardware and software setup was used:

Component	Specification
CPU	Intel Core i7 or AMD Ryzen 7 or higher
GPU	NVIDIA RTX 3060 or better (CUDA support)
RAM	Minimum 08 GB (32 GB recommended)
Storage	SSD with ≥250 GB
Display	Full HD (1920x1080) or higher
Internet	Required for downloading datasets & libraries

3.2 SoftwareRequirements

Software / Tool	Version / Description
Operating System	Windows 10/11
Python	Version 3.8+
Deep Learning Framework	TensorFlow 2.x and PyTorch 1.10+
Image Libraries	OpenCV, Pillow
Data Handling	NumPy, Pandas
Visualization Tools	Matplotlib, Seaborn
Development Environment	Jupyter Notebook / JupyterLab

Chapter 4
Proposed Methodology

4.1 Dataset Preparation

The proposed methodology involves designing, training, and evaluating convolutional neural networks (CNNs) for binary classification of post-hurricane building damage from satellite images. Two models are considered: a baseline CNN and a regularized CNN. Performance is evaluated using both balanced and unbalanced datasets, with a focus on improving generalization, accuracy, and interpretability.

The dataset used in this study comprises satellite images classified into two categories: damage and no_damage. The images are sourced from a custom dataset and pre-organized into the following splits:

- **Training Set**: 5000 images per class
- **Validation Set**: 1000 images per class
- **Unbalanced Test Set**: ~8000 images of damaged buildings and ~1000 of undamaged
- **Balanced Test Set**: 1000 images per class

Each image is resized to 128x128 pixels and read in RGB format. Class labels are encoded as binary values (0 for damage, 1 for no_damage) using LabelEncoder.

To visualize and confirm the distribution of class labels, count plots are generated using seaborn. RGB channel separation is also performed on sample images to understand the color composition of the satellite data.

4.1.1 Data Augmentation:

To improve generalization, the training and validation sets undergo augmentation with the following techniques using ImageDataGenerator:

- Rotation: ±10°
- Horizontal and vertical shift: ±20%
- Zoom: ±20%
- Horizontal flipping
- Brightness variation: 0.2 to 1.2
- Rescaling pixel values: 1/255

The test sets are not augmented to preserve the integrity of evaluation metrics.

4.2 Proposed Methodology Framework

The overall framework consists of the following components:

Step 1: Data Loading and Preprocessing

- Load images from directory structure.
- Resize and normalize pixel values.
- Encode categorical labels.
- Apply augmentation to training/validation datasets.

Step 2: Model Design

Two CNN architectures are proposed:

- **Baseline CNN**:
    - 3 Convolutional blocks (Conv2D + MaxPooling)
    - Fully connected dense layers
    - Sigmoid output for binary classification
- **Regularized CNN**:
    - Same structure as baseline
    - Additional BatchNormalization layers after each conv block
    - Dropout layers (5%) after dense layers

Step 3: Training

- Optimizer: Adam (learning rate = 0.0001)
- Loss Function: Binary Crossentropy
- Epochs: Baseline (10), Regularized (15)
- Training with validation monitoring

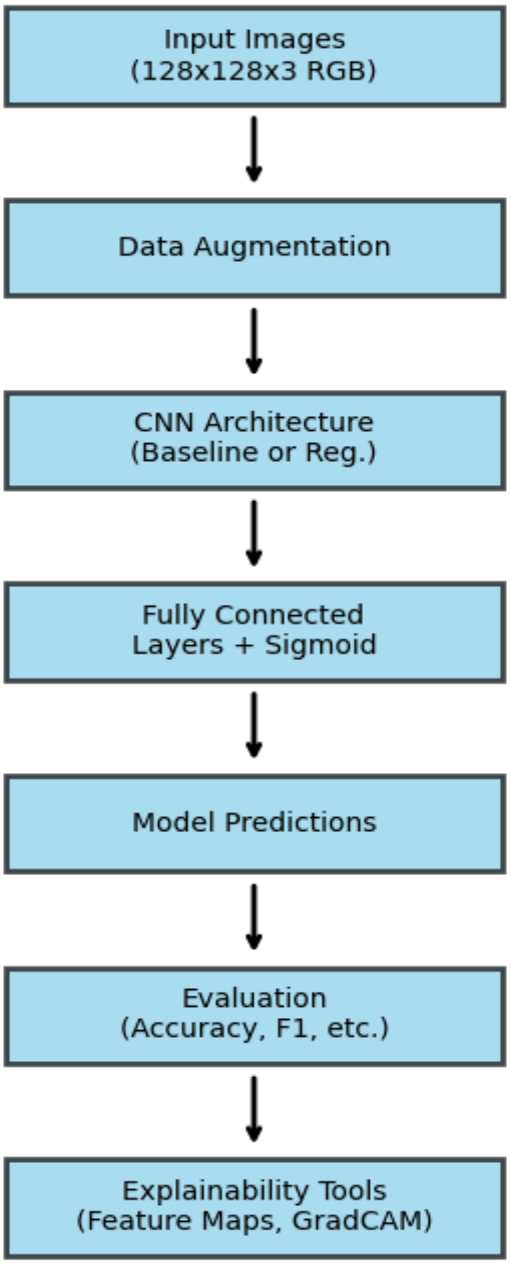
Step 4: Evaluation

- Evaluate both models on:
    - Balanced test dataset
    - Unbalanced test dataset
- Metrics: Accuracy, Precision, Recall, F1-Score, ROC AUC

Step 5: Interpretability

- Visualize feature maps from early convolutional layers.
- Use Grad-CAM and saliency maps (via tf-keras-vis) to interpret predictions and localize image regions influencing decisions.

Block Diagram – CNN-based Damage Detection Framework



4.3 Training Configuration

- **Input Shape**: (128, 128, 3)
- **Batch Size**: 100
- **Epochs**:
    - Baseline Model: 10
    - Regularized Model: 15
- **Loss Function**: Binary Crossentropy
- **Optimizer**: Adam with learning_rate = 1e-4
- **Metrics**: Accuracy, Precision, Recall, F1 Score, ROC AUC

4.4 Model Evaluation and Comparison

After training, both models are evaluated on two types of test sets:

- **Balanced Test Set**: Equal number of damage and no_damage images
- **Unbalanced Test Set**: Skewed toward damage images

Performance metrics (Accuracy, Precision, Recall, F1-Score, AUC) are calculated using scikit-learn to quantify model robustness and generalization. Visual comparisons of training/validation curves and evaluation scores are used to analyze overfitting, underfitting, and general trends.

Chapter 5
Results and Discussions

5.1 Evaluation Metrics

This section presents the evaluation results of the **regularized CNN model** on both unbalanced and balanced satellite image datasets for post-hurricane damage classification. The performance is discussed using metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **AUC**. The results are further interpreted to highlight model behavior and limitations.

- **Accuracy**: Overall proportion of correctly predicted samples.
- **Precision**: Measures the proportion of actual positives among all predicted positives.
- **Recall**: Measures the proportion of actual positives that were correctly identified.
- **F1-score**: Harmonic mean of precision and recall.
- **AUC (Area Under ROC Curve)**: Indicates the model's ability to distinguish between classes.

5.2 Performance on Unbalanced Dataset

Metric	Value
Accuracy	0.111
Precision	0.110
Recall	0.995
F1-score	0.199
AUC	0.498

5.2.1 Observations:

- The model shows **extremely high recall (0.995)** but **very low precision (0.110)**, suggesting that it predicted nearly all samples as the positive class, leading to many false positives.
- The **F1-score** is low due to the imbalance between precision and recall.
- **Accuracy (11%)** is far below random chance for a multi-class classification, indicating poor predictive power under unbalanced conditions.
- The **AUC score (0.498)** suggests that the model performs almost like random guessing in distinguishing between classes.

5.3 Performance on Balanced Dataset

Metric	Value
Accuracy	0.496
Precision	0.498
Recall	0.989
F1-score	0.662
AUC	0.496

5.3.1 Observations:

- The model performs **significantly better on the balanced dataset**. Accuracy is around **49.6%**, which, although modest, reflects a fair performance for a 4-class classification.

- **Precision (~0.498)** and **F1-score (~0.662)** are much higher than on the unbalanced dataset.

- The **recall remains very high (0.989)**, again suggesting over-prediction of positive classes.

- The **AUC (0.496)**, however, is still low and does not improve with balancing — possibly due to ineffective thresholding or confusion in multi-class boundaries.

5.4 Interpretation and Insights

- The extremely **high recall** paired with **low precision** implies the model is **biasing toward the majority class(es)**, which is common in softmax-based multi-class models when regularization and calibration are suboptimal.

- **Balanced data improves overall performance**, especially precision and F1-score, but further tuning is necessary to improve class discrimination (AUC).

- The low **AUC** scores in both datasets suggest the model struggles to form meaningful separation boundaries between classes — possibly a result of class similarities or lack of strong discriminatory features in the dataset.

5.5 Summary of Findings

- Regularization alone did not sufficiently address the bias introduced by class imbalance.

- While balancing improved most metrics (especially precision and F1), the model's discriminatory capacity remains weak as reflected by the low AUC.

- This analysis points to the need for further architectural or training improvements, such as:

    - **Focal loss** or **class-weighted loss functions**

    - **Data augmentation per class**

    - **More complex CNN variants or attention mechanisms**

    - **Better thresholding and calibration for multi-class probabilities**

Chapter 6
Conclusion

This project explored a deep learning-based approach for post-hurricane damage classification of buildings using satellite imagery. A convolutional neural network (CNN) architecture was implemented and further enhanced using regularization techniques. The model was evaluated on both unbalanced and balanced versions of the dataset to investigate the effects of class imbalance on classification performance.

The results clearly indicate that the **model struggled significantly on the unbalanced dataset**, with poor accuracy (11%), very low precision, and an AUC score near random chance. While recall remained high, this came at the cost of over-predicting positive classes and generating a large number of false positives.

In contrast, the **balanced dataset led to a noticeable improvement**, particularly in **precision** (from ~0.11 to ~0.50) and **F1-score** (from ~0.20 to ~0.66). This highlights the critical importance of **data balancing** in multi-class classification tasks where certain damage categories may be underrepresented. However, even with a balanced dataset and regularization applied, the **AUC scores remained low**, suggesting that the model still struggles to discriminate effectively between classes.

Overall, while the regularized CNN model achieved moderate performance under balanced conditions, its limitations in class separability and over-reliance on dominant features indicate the need for further model refinements. Future work could involve integrating **attention mechanisms**, **class-weighted or focal loss functions**, or **pre-trained architectures** to improve both interpretability and classification robustness.

Chapter 7
Future Work

While this project presents a deep learning-based approach for post-hurricane damage detection, there are several potential avenues for future enhancement and expansion. The following are key areas for future work:

1. **Class Imbalance Mitigation:**
   Explore advanced techniques like Focal Loss, Class-weighted Loss Functions, or Generative Adversarial Networks (GANs) for data augmentation to improve dataset balance and model performance.

2. **Advanced Architectures:**
   Investigate alternative CNN architectures (e.g., ResNet, DenseNet, EfficientNet) for better feature extraction and classification accuracy.
   Incorporate attention mechanisms (e.g., Self-Attention, Transformers) to focus on key regions like structurally damaged areas in images.

3. **Incorporating Multi-Temporal Data:**
   Utilize multi-temporal satellite imagery (pre- and post-event) to enhance damage classification. Use change detection algorithms to track structural changes over time for more accurate results.

4. **Transfer Learning:**
   Leverage pre-trained models (e.g., from ImageNet) and fine-tune for hurricane damage detection to improve performance with limited labeled data.

5. **Incorporating Additional Data Modalities:**
   Integrate multi-spectral or Synthetic Aperture Radar (SAR) data with optical images to improve detection under varying environmental conditions (e.g., cloud cover, different lighting).

6. **Real-time Monitoring and Deployment:**
   Develop real-time monitoring systems to automatically analyze incoming satellite images for timely alerts to disaster response teams, enabling quicker intervention during hurricanes or other natural disasters.

7. **Improved Explainability and Interpretability:**
   Enhance model interpretability with tools like LIME or SHAP to provide more transparent reasoning behind predictions, improving decision-making in high-stakes environments.

8. **Generalization to Other Disaster Types:**
   Extend models to detect damage from various disasters (e.g., floods, earthquakes, wildfires), improving the versatility and applicability of the model across different scenarios.

9. **Large-Scale Dataset Creation:**

Address the challenge of limited labeled data by developing a large-scale dataset of post-disaster satellite images, possibly through crowdsourcing or collaborations with governmental or humanitarian organizations.

10. **Automated Damage Severity Classification:**
Focus on damage severity classification to allow responders to prioritize regions with more severe damage, aiding in resource allocation and disaster management.

Chapter 8

References

Albawi, S., Mohammed, T. A. M., & Alzawi, S. (2017). Layers of a convolutional neural network. IEEE.

Betz, J. M., Brown, P. N., & Roman, M. C. (2011). Accuracy, precision, and reliability of chemical measurements in natural products research. Fitoterapia, 82(1), 44–52. https://doi.org/10.1016/j.fitote.2010.09.011

Cao, Q. D., & Choe, Y. (2020). Building damage annotation on post-hurricane satellite imagery based on convolutional neural networks. Natural Hazards, 103(3), 3357–3376. https://doi.org/10.1007/s11069-020-04133-2

Chen, S. A., Escay, A., Haberland, C., Schneider, T., Staneva, V., & Choe, Y. (2018). Benchmark dataset for automatic damaged building detection from post-hurricane remotely sensed imagery. arXiv. https://arxiv.org/abs/1812.05581

Dotel, S., Shrestha, A., Bhusal, A., Pathak, R., Shakya, A., & Panday, S. P. (2020). Disaster assessment from satellite imagery by analysing topographical features using deep learning. PervasiveHealth: Pervasive Computing Technologies for Healthcare, 86–92. https://doi.org/10.1145/3388818.3389160

Duarte, D., Nex, F., Kerle, N., & Vosselman, G. (2018). Satellite image classification of building damages using airborne and satellite image samples in a deep learning approach. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 4(2), 89–96. https://doi.org/10.5194/isprs-annals-IV-2-89-2018

Huynh, E., Hosny, A., Guthier, C., Bitterman, D. S., Petit, S. F., Haas-Kogan, D. A., Kann, B., Aerts, H. J. W. L., & Mak, R. H. (2020). Artificial intelligence in radiation oncology. Nature Reviews Clinical Oncology, 17(12), 771–781. https://doi.org/10.1038/s41571-020-0417-8

Kaur, S., Gupta, S., Singh, S., & Gupta, I. (2021). Detection of Alzheimer's disease using deep convolutional neural network. International Journal of Image and Graphics. https://doi.org/10.1142/S021946782140012X

Kovordányi, R., & Roy, C. (2009). Cyclone track forecasting based on satellite images using artificial neural networks. ISPRS Journal of Photogrammetry and Remote Sensing, 64(6), 513–521. https://doi.org/10.1016/j.isprsjprs.2009.03.002

Li, Y., Hu, W., Dong, H., & Zhang, X. (2019). Building damage detection from post-event aerial imagery using single shot multibox detector. Applied Sciences, 9(6), 1128. https://doi.org/10.3390/app9061128

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Han, J. (2019). On the variance of the adaptive learning rate and beyond. arXiv preprint. https://arxiv.org/abs/1908.03265

M, H., & M. N, S. (2015). A review on evaluation metrics for data classification evaluations. International Journal of Data Mining & Knowledge Management Process, 5(2), 1–11. https://doi.org/10.5121/ijdkp.2015.5201

Nia, K. R., & Mori, G. (2018). Building damage assessment using deep learning and ground-level image data. In Proceedings of the 14th Conference on Computer and Robot Vision (pp. 95–102). https://doi.org/10.1109/CRV.2017.54

Phung, V. H., & Rhee, E. J. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. Applied Sciences, 9(21), 4500. https://doi.org/10.3390/app9214500

Pi, Y., Nath, N. D., & Behzadan, A. H. (2020). Convolutional neural networks for object detection in aerial imagery for disaster response and recovery. Advanced Engineering Informatics, 43, 101009. https://doi.org/10.1016/j.aei.2019.101009

Pradhan, R., Aygun, R. S., Maskey, M., Ramachandran, R., & Cecil, D. J. (2018). Tropical cyclone intensity estimation using a deep convolutional neural network. IEEE Transactions on Image Processing, 27(2), 692–702. https://doi.org/10.1109/TIP.2017.2766358

Appendix I - Source Code

```
import numpy as np

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from sklearn.preprocessing import LabelEncoder

import os

import matplotlib.pyplot as plt

import tensorflow

from tensorflow.keras import Sequential,Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from PIL import Image

from pathlib import Path

from tensorflow.keras.utils import image_dataset_from_directory

import seaborn as sns

from tf_keras_vis.saliency import Saliency

from tf_keras_vis.gradcam import Gradcam

from tf_keras_vis.utils.scores import BinaryScore

from tf_keras_vis.utils.model_modifiers import ReplaceToLinear

from matplotlib import cm
```

```
#directories for train, validation and test images

train_directory = Path("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/train_another") # the training data; 5000 images of each class(damage/no damage)

validation_directory =
Path("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/validation_another") #the validation data; 1000 images of each class(damage/no damage)

unbalanced_test_directory =
Path("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/test_another") # 8000/1000 images of damaged/undamaged classes

balanced_test_directory = Path("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/test") # the balanced test data; 1000 images of each class(damage/no damage)
```

```
#Initialising empty lists for storing both of the damage and no_damage folders

labels_train,images_train,labels_test_another,images_test_another,labels_test,images_test = [],[],[],[],[],[]
```

```
for i in ['damage','no_damage']:
    files_train = os.listdir(f'C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/train_another/{i}')
    files_test_another =
os.listdir(f'C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/test_another/{i}')
    files_test = os.listdir(f'C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/test/{i}')
    images_train.append(files_train)
    images_test_another.append(files_test_another)
    images_test.append(files_test)

    for l in range(len(files_train)):
        labels_train.append(i)
    for l in range(len(files_test)):
        labels_test_another.append(i)
    for l in range(len(files_test_another)):
        labels_test.append(i)

print("Number of items in the training set : {}".format(len(labels_train)))

print("Number of items in the test_another dataset : {}".format(len(labels_test_another)))

print("Number of items in the test dataset : {}".format(len(labels_test)))

#Defining an instance of the LabelEncoder Class
label_en = LabelEncoder()
#Using label_en to transform the list
labels_all = [0]*3
labels_all[0] = label_en.fit_transform(labels_train)
labels_all[1] = label_en.transform(labels_test_another)
labels_all[2] = label_en.transform(labels_test)
fig,ax = plt.subplots(1,3,figsize = (15,5),sharey = True)
classes = ['damage','no_damage']
```

```
sns.countplot(x=list(labels_all[0]),ax = ax[0])

ax[0].set_title('For Training Dataset')

ax[0].set_xticks([0, 1]) # Ensure ticks match encoded labels

ax[0].set_xticklabels(classes)

# ax[0].set_ylabel('Number of Labels for each class')

sns.countplot(x=labels_all[1],ax = ax[1])

ax[1].set_xticks([0, 1])

ax[1].set_title('For Balanced Test Dataset')

# ax[1].set_ylabel('Number of Labels for each class')

ax[1].set_xticklabels(classes)

sns.countplot(x=labels_all[2],ax = ax[2])

ax[2].set_title('For Unbalanced Test Dataset')

# ax[2].set_ylabel('Number of Labels for each class')

ax[2].set_xticks([0, 1])

ax[2].set_xticklabels(classes)

plt.show()



import numpy as np


# Check unique values in labels

for i in range(3):

    print(f"Unique values in labels_all[{i}]:", np.unique(labels_all[i]))

#quick image display with PIL

img1 =
Image.open("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/train_another/damage/-93.66975_30
.218802.jpeg") #train_damaged

plt.imshow(img1)

plt.title(f'Actual label = {labels_train[0]}')

plt.axis('off')

plt.show()
```

```
# Turn our image object into a NumPy array
img_data = np.array(img1)

# get the shape of the resulting array
img_data_shape = img_data.shape

print("Our NumPy array has the shape: {}".format(img_data_shape))

# plot the data with `
`
fig,ax = plt.subplots(1,5,figsize=(20,20),sharey=True)

ax[0].imshow(img_data)
ax[0].axis('off')
# grayscale image
ax[1].imshow(img_data[:,:,2], cmap='gray',interpolation='nearest')
ax[1].axis('off')
# plot the red channel
ax[2].imshow(img_data[:,:,0], cmap=plt.cm.Reds_r)
ax[2].axis('off')
# plot the green channel
ax[3].imshow(img_data[:,:,1], cmap=plt.cm.Greens_r)
ax[3].axis('off')

# plot the blue channel
ax[4].imshow(img_data[:,:,2], cmap=plt.cm.Blues_r)
ax[4].axis('off')

plt.show()
train_gen = ImageDataGenerator(rotation_range=10, # rotation
      width_shift_range=0.2, # horizontal shift
```

```
        height_shift_range=0.2, # vertical shift
        zoom_range=0.2, # zoom
        horizontal_flip=True,# horizontal flip
        rescale=1/255.0, #re-scaling
        brightness_range=[0.2,1.2]) # brightness)
validation_gen = ImageDataGenerator(rotation_range=10, # rotation
        width_shift_range=0.2, # horizontal shift
        height_shift_range=0.2, # vertical shift
        zoom_range=0.2, # zoom
        horizontal_flip=True, # horizontal flip
        rescale=1/255.0, #re-scaling
        brightness_range=[0.2,1.2]) # brightness)
test_unbalanced_gen = ImageDataGenerator()
test_balanced_gen = ImageDataGenerator()train_data = train_gen.flow_from_directory(
    directory = train_directory,
    target_size = (128,128),
    class_mode = 'binary',
    color_mode='rgb',
    shuffle = True,
    batch_size=100)

val_data = validation_gen.flow_from_directory(
    directory = validation_directory,
    target_size = (128,128),
    class_mode = 'binary',
    color_mode='rgb',
    shuffle = True,
    batch_size=100)
unbalanced_data = test_unbalanced_gen.flow_from_directory(directory =unbalanced_test_directory,
                                          target_size = (128,128),
                                          class_mode = 'binary',
```

```
                                        shuffle = False,
                                        color_mode='rgb',
                                        batch_size=100)

balanced_data = test_balanced_gen.flow_from_directory(directory =balanced_test_directory,
                                        target_size = (128,128),
                                        class_mode = 'binary',
                                        color_mode='rgb',
                                        shuffle =False,
                                        batch_size=100)
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Flatten,BatchNormalization,Dropout

# Initialize a sequential model
model = Sequential(name="Base_Model")

model.add(tensorflow.keras.layers.Input(shape=(128, 128, 3)))
model.add(Conv2D(32,kernel_size =(3, 3), activation='relu'))
model.add(Conv2D(32,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model.add(Conv2D(64,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model.add(Conv2D(128,kernel_size =(3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))
model.add(Dense(1024,activation='relu'))
```

```
model.add(Dense(1,activation = 'sigmoid'))

# tensorflow.keras.utils.plot_model(model, "base_model.png", show_shapes=True,dpi =50)

# Compiling the model

model.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate = 1e-4),
loss='binary_crossentropy', metrics=['accuracy'])

history_1 = model.fit(train_data,validation_data=val_data,epochs=10)

model.save("normal_model")

from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

def metrices(model) :

    y_preds_unbalanced = [1 if prob >= 0.5 else 0 for prob in model.predict(unbalanced_data)]

    y_preds_balanced = [1 if prob >= 0.5 else 0 for prob in model.predict(balanced_data)]

    y_true_unbalanced = unbalanced_data.classes

    y_true_balanced = balanced_data.classes



    precision_unbalanced = precision_score(y_true_unbalanced, y_preds_unbalanced)

    recall_unbalanced = recall_score(y_true_unbalanced, y_preds_unbalanced)

    f1_unbalanced = f1_score(y_true_unbalanced, y_preds_unbalanced)

    roc_auc_unbalanced= roc_auc_score(y_true_unbalanced, y_preds_unbalanced)


    precision_balanced = precision_score(y_true_balanced, y_preds_balanced)

    recall_balanced = recall_score(y_true_balanced, y_preds_balanced)

    f1_balanced = f1_score(y_true_balanced, y_preds_balanced)

    roc_auc_balanced = roc_auc_score(y_true_balanced, y_preds_balanced)


    print('Precision on Unbalanced Test Dataset : {}'.format(precision_unbalanced ))

    print('Precision on Balanced Test Dataset : {}'.format(precision_balanced))


    print('Recall on Unbalanced Test Dataset : {}'.format(recall_unbalanced))

    print('Recall on Balanced Test Dataset : {}'.format(recall_balanced))


    print('F1 Score on Unbalanced Test Dataset : {}'.format(f1_unbalanced))
```

```
    print('F1 Score on Balanced Test Dataset : {}'.format(f1_balanced))


    print('AUC score on Unbalanced Test Dataset : {}'.format(roc_auc_unbalanced))

    print('AUC score on Balanced Test Dataset : {}'.format(roc_auc_balanced))


    return precision_unbalanced,precision_balanced,
recall_unbalanced,recall_balanced,f1_unbalanced,f1_balanced ,roc_auc_unbalanced,roc_auc_balanced


#plot of train loss & validation loss for the base model


fig, ax = plt.subplots(1,2,figsize = (25,5))

ax[0].plot(history_1.history['loss'])

ax[0].plot(history_1.history['val_loss'])

ax[0].legend(['train', 'validation'])

ax[0].set_xlabel('epoch')

ax[0].set_ylabel('Loss')

ax[0].set_title('Train & Validation Losses for Base model')


#plot of train accuracy & validation accuracy for the base model


ax[1].plot(history_1.history['accuracy'])

ax[1].plot(history_1.history['val_accuracy'])

ax[1].legend(['train', 'validation'])

ax[1].set_xlabel('epoch')

ax[1].set_ylabel('accuracy')

ax[1].set_title('Train & Validation accuracies for Base model')


plt.show()

test_unbalanced_acc_1 = model.evaluate(unbalanced_data,verbose=0)

print('Test Accuracy of Specified Base Model on Unbalanced Test Dataset :
{}'.format(test_unbalanced_acc_1[1] ))

test_balanced_acc_1= model.evaluate(balanced_data,verbose =0)
```

```
print('Test Accuracy of Specified Base Model on Balanced Test Dataset : {}'.format(test_balanced_acc_1[1]
))



metrics_1 = metrices(model)



val_img1 =
Image.open("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/validation_another/damage/-93.955
831_30.1334.jpeg")

val_img2 =
Image.open("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/validation_another/damage/-93.799
735_30.036396000000003.jpeg")

val_img3 =
Image.open("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/validation_another/no_damage/-95.
067337_29.831393.jpeg")

val_img4 =
Image.open("C:/Users/deepa/Desktop/jupyter-pr/DL_PROJECT/test_2/validation_another/no_damage/-95.
64264_29.844161.jpeg")

val_img_labels = [0,0,1,1]

val_imgs = [val_img1,val_img2,val_img3,val_img4]

val_img_data = []

for img_data in val_imgs:

    data = np.array(img_data)

    data = data/255.0

    data = np.expand_dims(data,axis=0)

    #print(data.shape)

    val_img_data.append(data)

#Helper function used to plot the activation maps

def plot_featuremaps(img,activations,layer_names):

    fig, axs = plt.subplots(ncols=4, nrows=4,figsize = (6,6))

    gs = axs[1, 2].get_gridspec()

    # remove the underlying axes

    for ax in axs[1:-1, 1:-1]:

        ax[0].remove()

        ax[1].remove()

    axbig = fig.add_subplot(gs[1:-1, 1:-1])
```

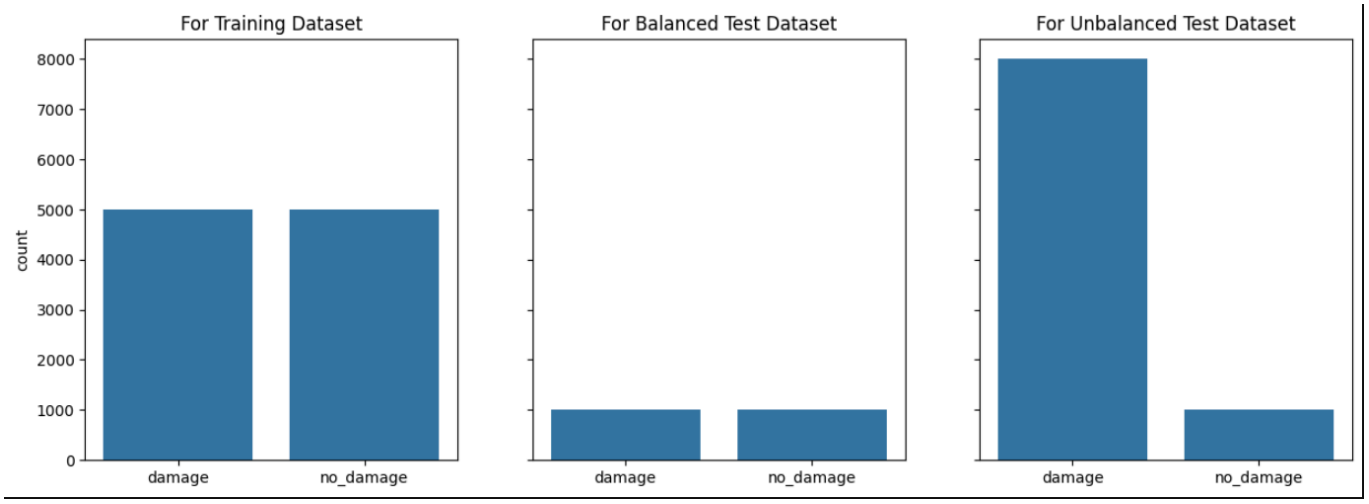
```
    axbig.imshow(img.squeeze())
    axbig.axis('off')

    for i, axis in enumerate(axs.ravel()):
        axis.imshow(activations.squeeze()[:,:,i])
        axis.axis('off')

    fig.tight_layout()
    fig.suptitle(f'Feature maps for {layer_names[0]}',y=1.05);

first_conv_layer_output = model.layers[0].output
activation_model = Model(inputs=model.layers[0].input,outputs=first_conv_layer_output)
activations = activation_model.predict(val_img_data[0])
plot_featuremaps(val_img_data[0],activations,[model.layers[0].name])
#Defining score function for saliency maps and GradCAM heatmap
def score_function(index):
    if (index==0 or index==1):
        score = BinaryScore(0.0)
    else:
        score = BinaryScore(1.0)
    return score
from tensorflow.keras.layers import Conv2D,MaxPool2D,Dense,Flatten
from tensorflow.keras import regularizers
# Initialize a sequential model
model_reg = Sequential(name="Regularized_Model")


model_reg.add(Conv2D(32,kernel_size =(3, 3), activation='relu',input_shape=(128,128,3)))
model_reg.add(Conv2D(32,kernel_size =(3,3), activation='relu'))
model_reg.add(BatchNormalization())
```

```
model_reg.add(MaxPool2D(pool_size=(2, 2)))

model_reg.add(Conv2D(64,kernel_size =(3,3), activation='relu'))

model_reg.add(Conv2D(64,kernel_size =(3,3), activation='relu'))

model_reg.add(BatchNormalization())

model_reg.add(MaxPool2D(pool_size=(2, 2)))

model_reg.add(Conv2D(128,kernel_size =(3,3), activation='relu'))

model_reg.add(Conv2D(128,kernel_size =(3,3), activation='relu'))

model_reg.add(BatchNormalization())

model_reg.add(MaxPool2D(pool_size=(2, 2)))


model_reg.add(Flatten())


model_reg.add(Dense(1024,activation='relu'))

model_reg.add(Dropout(0.05))

model_reg.add(Dense(1024,activation='relu'))

model_reg.add(Dropout(0.05))


model_reg.add(Dense(1,activation = 'sigmoid'))

# Compiling the model

model_reg.compile(optimizer=tensorflow.keras.optimizers.Adam(learning_rate = 1e-4),
loss='binary_crossentropy', metrics=['accuracy'])

history_2 =
model_reg.fit(train_data,validation_data=val_data,epochs=15)model_reg.save("regularised_model")

model_reg.save("regularised_model")

#Accuracy plots of Regularized and base model


fig, ax = plt.subplots(1,2,figsize = (25,5))

ax[0].plot(history_1.history['accuracy'])

ax[0].plot(history_1.history['val_accuracy'])

ax[0].legend(['train', 'validation'])

ax[0].set_xlabel('epoch')

ax[0].set_ylabel('accuracy')
```

```
ax[0].set_title('model accuracy for Base')


ax[1].plot(history_2.history['accuracy'])

ax[1].plot(history_2.history['val_accuracy'])

ax[1].legend(['train', 'validation'])

ax[1].set_xlabel('epoch')

ax[1].set_ylabel('accuracy')

ax[1].set_title('model accuracy for Regularized')




plt.show()


test_unbalanced_acc_2 = model_reg.evaluate(unbalanced_data,verbose=0)

print('Test Accuracy of Regularised Model on Unbalanced Test Dataset :
{}'.format(test_unbalanced_acc_2[1] ))

test_balanced_acc_2= model_reg.evaluate(balanced_data,verbose =0)

print('Test Accuracy of Regularised Model on Balanced Test Dataset : {}'.format(test_balanced_acc_2[1] ))


metrics_2 = metrices(model_reg)
```

Appendix II - Screenshots

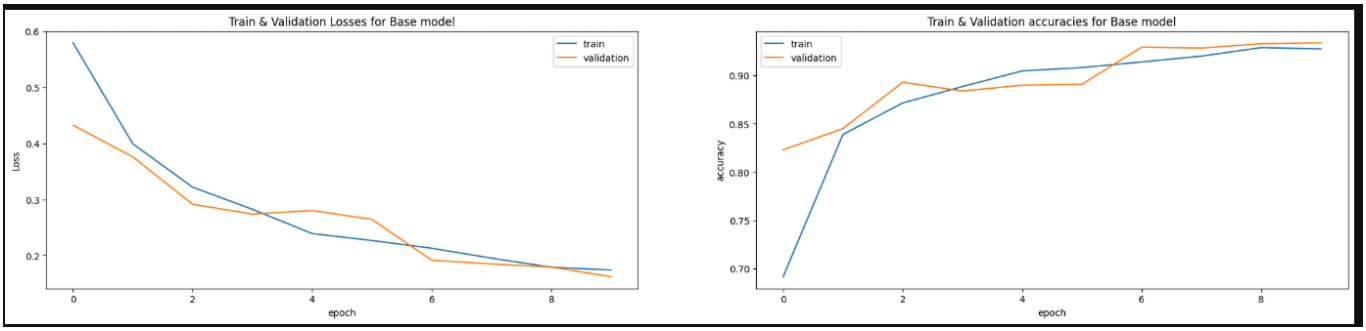
Dataset Distribution



Training

```
Epoch 1/10
100/100 [==============================] - 171s 1s/step - loss: 0.5794 - accuracy: 0.6915 - val_loss: 0.4324 - val_accuracy: 0.8230
Epoch 2/10
100/100 [==============================] - 116s 1s/step - loss: 0.3992 - accuracy: 0.8389 - val_loss: 0.3759 - val_accuracy: 0.8450
Epoch 3/10
100/100 [==============================] - 120s 1s/step - loss: 0.3220 - accuracy: 0.8716 - val_loss: 0.2915 - val_accuracy: 0.8930
Epoch 4/10
100/100 [==============================] - 121s 1s/step - loss: 0.2825 - accuracy: 0.8886 - val_loss: 0.2739 - val_accuracy: 0.8840
Epoch 5/10
100/100 [==============================] - 120s 1s/step - loss: 0.2392 - accuracy: 0.9048 - val_loss: 0.2802 - val_accuracy: 0.8900
Epoch 6/10
100/100 [==============================] - 127s 1s/step - loss: 0.2268 - accuracy: 0.9083 - val_loss: 0.2646 - val_accuracy: 0.8910
Epoch 7/10
100/100 [==============================] - 119s 1s/step - loss: 0.2130 - accuracy: 0.9141 - val_loss: 0.1916 - val_accuracy: 0.9295
Epoch 8/10
100/100 [==============================] - 120s 1s/step - loss: 0.1953 - accuracy: 0.9202 - val_loss: 0.1851 - val_accuracy: 0.9285
Epoch 9/10
100/100 [==============================] - 119s 1s/step - loss: 0.1792 - accuracy: 0.9290 - val_loss: 0.1795 - val_accuracy: 0.9330
Epoch 10/10
100/100 [==============================] - 125s 1s/step - loss: 0.1741 - accuracy: 0.9276 - val_loss: 0.1622 - val_accuracy: 0.9340
```

Loss & accuracy




```
Test Accuracy of Specified Base Model on Unbalanced Test Dataset : 0.964555561542511
Test Accuracy of Specified Base Model on Balanced Test Dataset : 0.8939999938011169
Precision on Unbalanced Test Dataset : 0.857292759706191
Precision on Balanced Test Dataset : 0.9769975786924939
Recall on Unbalanced Test Dataset : 0.817
Recall on Balanced Test Dataset : 0.807
F1 Score on Unbalanced Test Dataset : 0.8366615463389657
F1 Score on Balanced Test Dataset : 0.8838992332968236
AUC score  on Unbalanced Test Dataset : 0.9
AUC score on Balanced Test Dataset : 0.894
```

Testing accuracy

```python
loss, accuracy = model.evaluate(test_dataset)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.2%}")
```

```
63/63 [==============================] - 5s 42ms/step - loss: 14.2723 - accuracy: 0.9010
Test Loss: 14.2723
Test Accuracy: 90.10%
```