Image Classification Structural Building Report

Sean Hershey, Nipun Das, and Alejandra Bravo
Cal Poly,
San Luis Obispo, United States
{shershey,nkdas,abravo03}@calpoly.edu

***Abstract*—Structural building damage comes in many different forms and levels, and can be classified by using images of the damage. Our aim was to use CNNs to classify the level of damage depicted in a picture. Our approach has improved extensibility and accuracy of structural damage classification with a CNN. The main criteria we analyzed our model on is accuracy with a secondary goal of computational efficiency.**

I. Introduction

Image classification was implemented to detect the level of structural damage of buildings. We evaluated the damage level found in images using an open-sourced structural image database [1] that classifies the damage level into one of four categories: Undamaged, Minor Damage, Moderate Damage, and Heavy Damage.

Structural analysis is often done by reconnaissance teams who require civil engineering backgrounds and extensive time in the field to correctly classify structural damage. In humanitarian relief situations following natural disasters like earthquakes, time is of the essence in delivering aid to the most affected areas. This is especially important in undeserved areas where there may not be easy access to surveyors who can analyze structural damage. By analyzing structural damage from just images with a Convolutional Neural Network (CNN), there is potential we could open up better capability in emergency response and research.

The goals of our approach were improved extensibility and accuracy of structural damage classification.Our implementation is outlines as follows:

- created input pipeline
- built CNN model using TensorFlow [2] and Keras API
- tested model
- improved accuracy and corrected overfitting by modifying model parameters
- created a demonstration using Gradio

Our final model reached a training accuracy of 68.3% and a test accuracy of 60.2% with 100 epochs.

II. Specification

A. Inputs

The only inputs used for our work were dataset files downloaded from the PEER (Pacific Earthquake Engineering Research Center) Hub ImageNet (PHI-Net or Φ-Net) [1]. The Φ-Net framework is an open-sourced structural image database, and there are currently eight benchmark classification tasks:

1) three-class classification for scene level
2) binary classification for damage state
3) binary classification for spalling condition (material loss)
4) binary classification for material type
5) three-class classification for collapse mode
6) four-class classification for component type
7) **four-class classification for damage level**
8) four-class classification for damage type

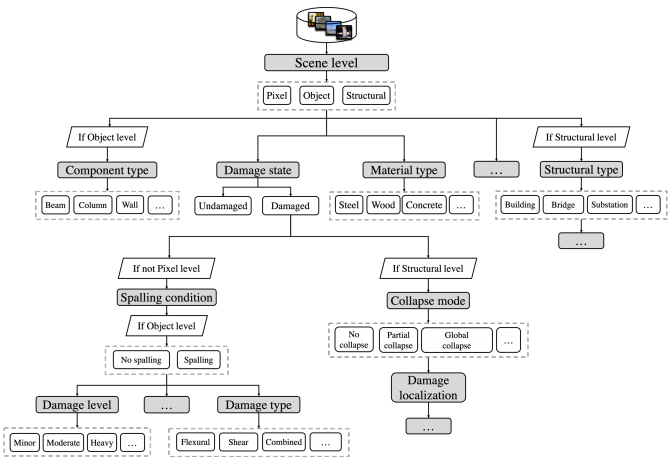

Fig. 1. The proposed framework with a hierarchy-tree structure, where grey boxes represent the detection tasks for the corresponding attributes, white boxes within the dashed lines are possible labels of the attributes to be chosen in each task, and ellipsis in boxes represent other choices or conditions.

The data set we used is task 7 four-class classification for damage level and includes a label or classification for each image corresponding to damage level: undamaged, minor, moderate, and heavy damage seen in Figure 4 [3]. The data set is labeled by humans as seen in the acknowledgments section of the data set. The composition or balance of the data set is as follows: 38% undamaged, 19% minor damage, 21% moderate damage, and 22% heavy damage as seen in Figure 2. The dataset had 88% training images and 12% test validation images shown in Figure 3. This is relatively well balanced and yielded a tolerably evenly trained model.

- task7_X_test.npy
- task7_X_train.npy
- task7_y_test.npy
- task7_y_train.npy

The NumPy array files included 4,138 bitmap images of 224x224 px dimensions. This worked out to be a good size

for our CNN and would not need pre-proccess or in-sequence scaling.

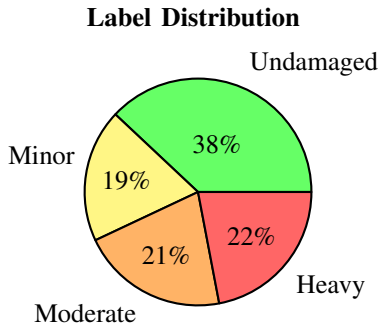

Fig. 2. The distribution of label frequency in the dataset.

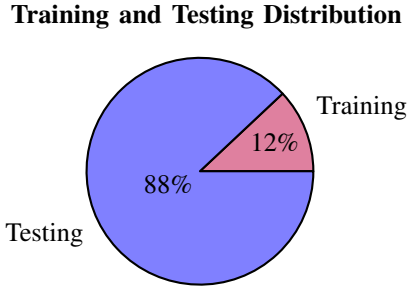

Fig. 3. The distribution of training and testing images in the dataset.

B. Outputs

The outputs of our project were a trained CNN model, the results of the image classification, model validation metrics, and a demonstration using Gradio for a GUI.

C. Parameters

The parameters we investigated in regard to the CNN model were: activation function, dropout rate, and optimizer.

III. Related Work

Detecting structural damage with image classification has been researched in the past. A 2017 paper researching this topic [4] discusses the use of Convolutional Neural Networks in detecting damage level and other characteristics of structural buildings from images, since CNNs work particularly well for machine learning tasks involving images [2]. This paper also discusses the use of a technique called Transfer Learning, which saves computational time and resources by picking up on features from other already trained CNNs.

The research also focused on the tweaking of CNNs to classify structural building damage at a higher accuracy. experimenting with aspects of the model like the number of convolutional layers or the number of neurons in each layer, analyzing their impact on the performance of the model. Other techniques mentioned in the Deep Transfer Learning paper [4] include using feature extraction to reduce the number of convolutional layers needed in the network, and using dropout layers to help reduce overfitting of the model. The Deep Transfer Learning paper had approximately 2000 labeled images, which was considered low and required mitigation to prevent ”severe overfitting” [4]. Since our data set has 4138 images, overfitting was an issue that we accounted for as well, as discussed later.

IV. Implementation

From reading references, we have found that the damage scale of image identification affects the results and insights. Satellite data or drone data gives insights on total area damage whereas different levels of zoom on the structural components of a structure can give insights into damage on an object level. Our detection task involves analyzing damage on this object level. The images we will use include close up views of walls or pillars, as an example.

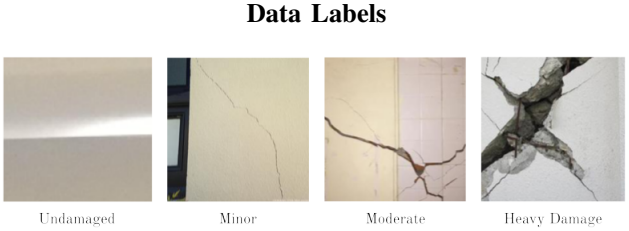

Fig. 4. Examples of the images corresponding to each of the data labels

An input pipeline allowed us to load our data set, where the Keras utility [2] allowed for streamlining. Then we built and configured the model using a CNN. Finally we trained the model and test it. The included Keras API allowed us to debug and develop in Python and actively monitor the training process throughout and make changes to our proccess as needed. The Keras API also provided us with a simple way to adjust details of our model like the number of neurons in each layer or the number of layers in our final model. Other options exist for model framework libraries like Torch or Caffe but the ease of setup and documentation for TensorFlow and Keras aided in our choice to use it.

Data augmentation was necessary as loss and accuracy of the training and test data diverged significantly after just 20 epochs, as seen in Figure 5. We added a dropout layer to help jolt the training in promising directions by nullifying out random neurons at a chance of 20%. In addition we also augmented the data by applying the following probabilistic modifications.

- horizontal flip
- vertical flip
- rotation up to 30 degrees
- zoom up to 30%

In terms of the specific details of our CNN model, we used TensorFlow’s documentation on CNNs [2], as well as the experimentation from the Deep Transfer Learning paper [4], as a starting point for the creation of our own CNN. From there, we adjusted and added layers to improve accuracy and

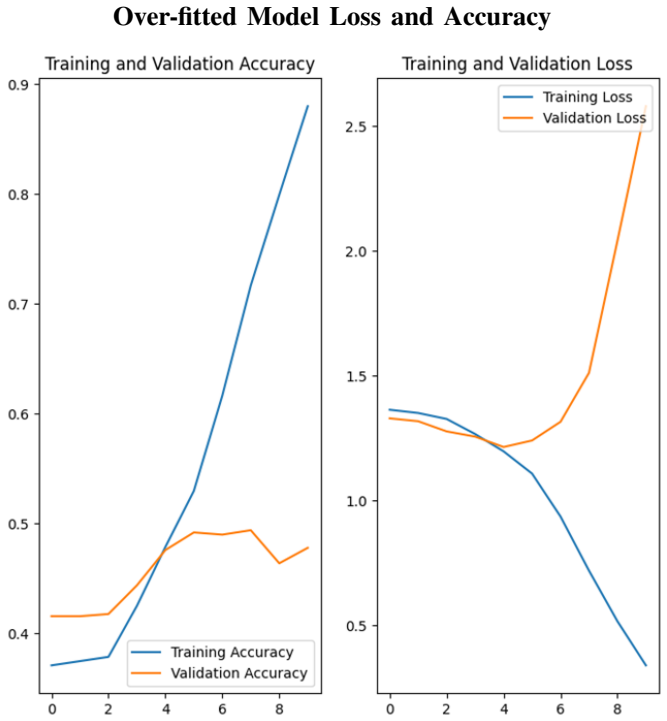

Fig. 5. Our initial attempt at training a CNN model, demonstrating over-fitting of the model on the training set. The plateauing of the test accuracy while training accuracy continues to increase is the indicator of over-fitting.

improve computational cost (as a secondary goal). The Layers broke down as follows.

1) Conv2D 16 relu
2) MaxPooling2D
3) Conv2D 32 relu
4) MaxPooling2D
5) Conv2D 64 relu
6) MaxPooling2D
7) Conv2D 128 relu
8) MaxPooling2D
9) Dropout 0.2
10) Flatten
11) Dense 128 relu
12) Dense 4 softmax

For our demo, we took the saved .tflite model file and loaded it using Tensorflow's tflite interpreter. Then, using Gradio, we were able to easily create an interface that takes images as inputs, resizes and converts them to the 224x224x3 numpy array format that our model takes as input, and then provide the confidence for the model. A screenshot of the demo interface and what a possible input/output would look like when processed through our model is shown in Figure 6.

V. Evaluation / Results

The baseline we are comparing our model performance to is the deep transfer learning model [4]. We are primarily evaluating results by accuracy of the test data set, since we never measured other aspects of the model like speed.

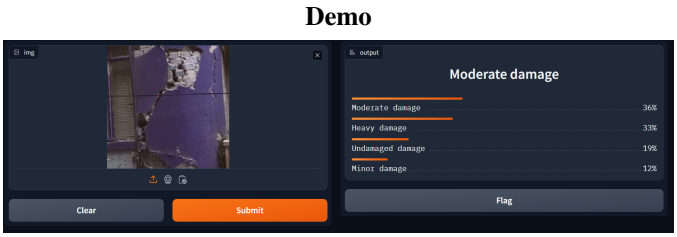

Fig. 6. A screenshot of the Gradio demo interface that allows for testing of the model on uploaded image files.

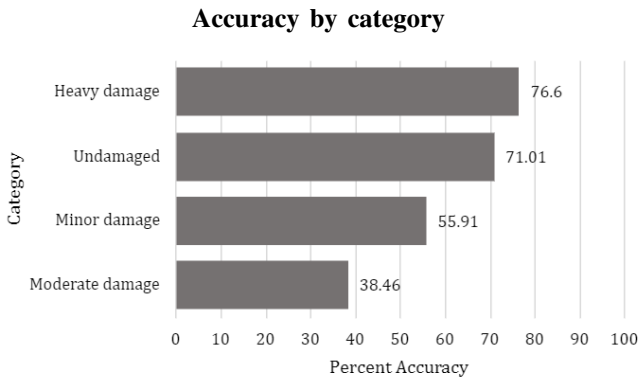

Fig. 7. The test validation accuracy by category of the model.

Importantly, the deep transfer learning model [4] uses complex techniques like Deep Transfer learning on top of a CNN structure with more layers and more neurons, so our goal is not necessarily to match the accuracy achieved by the deep transfer learning model.

The deep transfer learning model achieved a training accuracy 91.8% and a test accuracy of 68.8%. Our model, by comparison, achieved a training accuracy of 68.3% and a test accuracy of 60.2%. Our final model was achieved after 100 epochs of training on the test dataset, figures of which can be seen in Figure 8.

It achieved various levels of accuracy depending on the label or category of damage as seen in Figure 7. It was most confident in the Heavy Damage category as even to us these we often the most obvious so it makes sense the model would come to this determination as well despite undamaged being the most represented in the training data set. Undamaged was second while moderate damage was the least confidence inspiring for the model. This can be explained by moderate having the least representation in the training dataset and undamaged having the most.

Overall, the the performance of our model is fairly good given the constraints we had. We ultimately used a simple CNN model from TensorFlow with no other techniques besides those used to reduce over-fitting, so being able to achieve test accuracy almost as high as the deep transfer learning model is reasonable. Furthermore, Figure 8 demonstrates that we were able to avoid over-fitting in our model.

The data set from the ImageNet challenge provides both a training and a testing data set, allowing us to evaluate

Model Accuracy and Loss

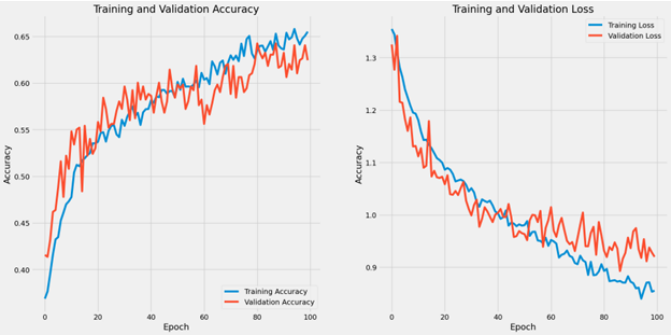

Fig. 8. A graph of model accuracy on the left and loss on the right. Red lines are for the training set and blue lines are for the validation set.

each iteration our models after completing training to measure accuracy, our primary measure to be optimized. Additionally, we may seek out other image data sets of object-level structural damage to help ensure that our model does not suffer from overfitting.

VI. Summary

Our initial goal was to train a CNN model to use image classification to detect the level of structural damage of buildings. Ultimately, we were able to create a model that accomplished this with accuracy comparable to already-existing solutions to this problem.

There are a variety of next steps that could be taken in this project. For one, more experimentation should be done with the structure of the model (number of convolutional/pooling layers or number of neurons in densely connected layers). Since we spent most of our time getting the TensorFlow environment set up and then remedying over-fitting issues, this is an aspect of experimentation that we did not get to within the time frame of the project.

Another area of experimentation would be with the number of epochs to train the model for. The deep transfer learning model had some separation between the training and test accuracy [4], something which we were trying to minimize to avoid over-fitting. Although that was probably a good instinct given our over-fitting issues on the first pass of training the model, some separation between training and test accuracy may not necessarily be a bad thing, and could potentially lead to improvements in accuracy while still avoiding over-fitting.

Finally, model performance is something that could be analyzed more, and possibly optimized for. Given that our model has significantly fewer neurons in the densely connected layers compared to the deep transfer learning model, we believe that our model would require less computational resources and would perform faster. However, being able to properly measure that would justify those beliefs. Additionally, this would open the possibility of optimizing the model to balance performance and accuracy, rather than only account for accuracy like the initial goal of this project was currently.

Ultimately, a CNN model allows us to do damage level evaluation of pixel and object level images on a data set from [1]. This technology is critical in humanitarian relief situations following natural disasters like earthquakes time is of the essence in delivering aid to the most affected areas.

References

[1] "Peer hub image-net," 2018, access: 01/31/2024.
[2] TensorFlow, "Module: tf," https://www.tensorflow.org/api_docs/python/tf, 2023, access: 02/02/2024.
[3] . M. K. Gao Y., "Peer hub imagenet (-net): A large-scale multi-attribute benchmark dataset of structural images," 2019.
[4] Y. Gao and K. Mosalam, "Deep transfer learning for image-based structural damage recognition," *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, 04 2018.
[5] IEEE, "Manuscript templates for conference proceedings," https://www.ieee.org/conferences/publishing/templates.html, 2019, access: 02/02/2024.

The template for this document is from [5].