Springboard – Data Science Career Track
Capstone Project 3

Identification and classification of damaged/flooded buildings from post-hurricane satellite images

Biniam Asmerom

April, 2022

Table of Contents

1. INTRODUCTION **2**

2. DATA **2**

3. EXPLORATORY DATA ANALYSIS (EDA) **4**

3.1. Principal Component Analysis (PCA) 4

4. MODELING **5**

4.1. Computer vision 5
4.2. Transfer learning 6
4.3. VGG16 model 7
4.4. Performance Metrics 8
4.5. Base Line Model 8
4.6. Extended Modeling 10

5. FINDINGS **15**

6. CONCLUSION **18**

7. FUTURE WORKS **18**

8. RECOMMENDATION **19**

9. RESOURCES **20**

1. Introduction

After a hurricane, damage assessment is critical to emergency managers for efficient response and resource allocation. One way to gauge the damage extent is to quantify the number of flooded/damaged buildings, which is traditionally done by ground survey. This process can be labor-intensive, time-consuming as well as some areas can be inaccessible due to flooding.

Another way to assess hurricane damage level is flood detection through synthetic aperture radar (SAR) images. SAR imagery is useful in terms of mapping different surface features, texture, or roughness pattern but is harder for laymen to interpret than optical sensor imagery. The resolutions of virtually all SAR images of today are too coarse to permit the building-level (as opposed to regional-level) damage assessment.

Recently, imagery taken from drones and satellites started to help improve situational awareness from a bird's eye view, but the process still relies on human visual inspection of captured imagery, which is generally time-consuming and unreliable during an evolving disaster. An automated damage/flood detection of buildings would be very beneficial. The main objective of this project is therefore, to identify and classify damaged building using computer vision techniques.

2. Data

Dataset for this project is a satellite imagery data of the Greater Houston area before and after Hurricane Harvey in 2017 (Figure 1). The images are labeled as either damaged or undamaged. The datasets are downloaded from Kaggle [1] and they are grouped into 4 groups as described below (see Figure 2):

Training data*:* the training data contains 5000 images of each class
validation: the validation data with 1000 images of each class
test: balanced test data with 1000 images of each class
test-another: unbalanced test data with 8000/1000 images of damaged/undamaged classes

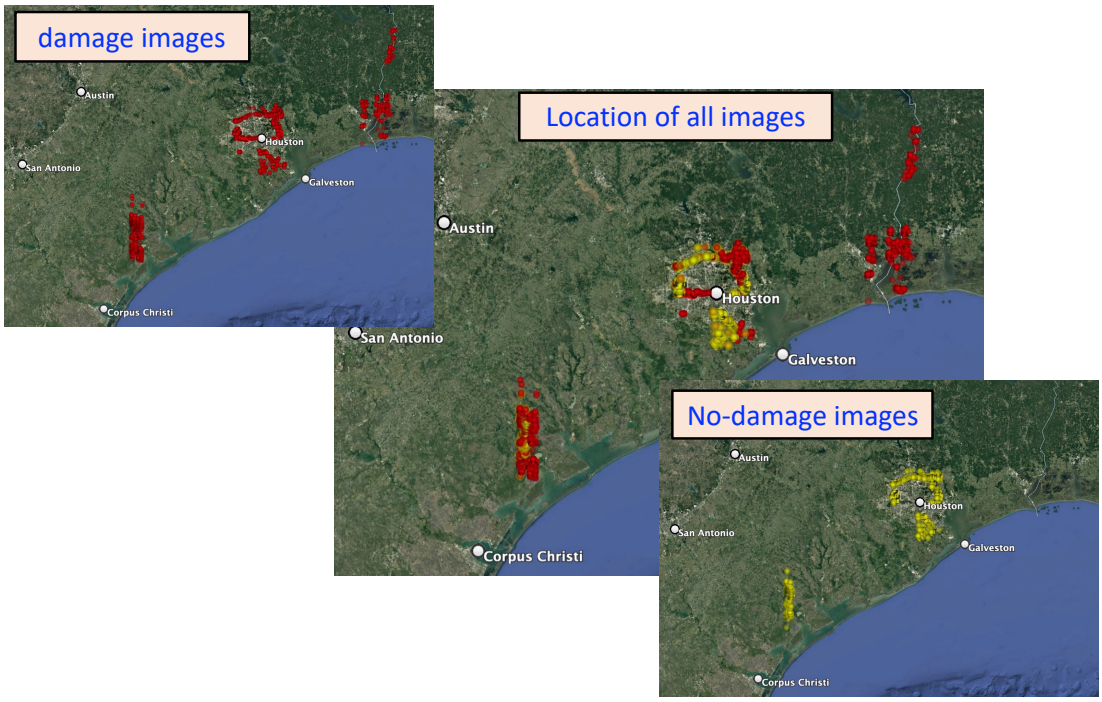


Figure 1 Location map of input datasets

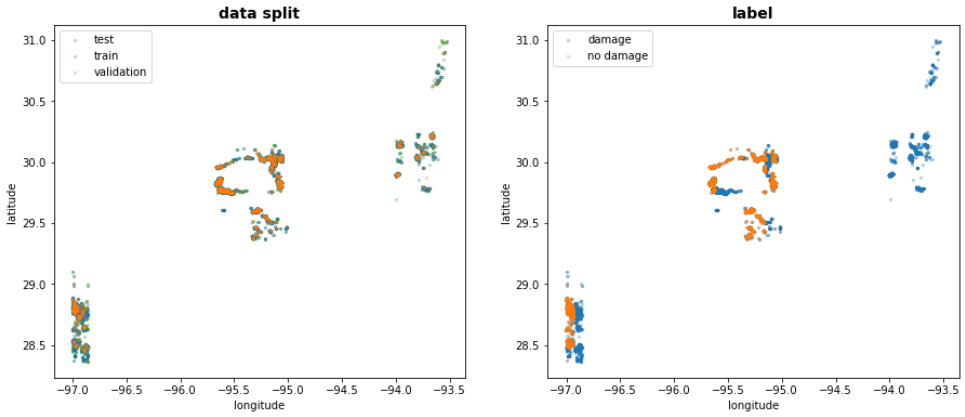


Figure 2 Distribution of images by data type (data-split)

From the map in figure 1 and 2, we can see that the datasets are gathered from roughly 4 clusters. The areas include the greater Houston area, Victoria, Beaumont and right at the boundary of the Newton and Beauregard counties in Texas and Louisiana respectively.

Observations from the above plots:

- The samples from the Beaumont and the border of Newton County are only damage class, while the other locations have both classes.
- Even in the locations where we have mix of both damage and no-damage samples, there seems to be a clear boundary b/n samples of damage and no-damage. This is mostly surprising as flooding always affects entire neighborhoods.
- The training, validation and test sample all the locations very well, meaning they are distributed in all locations.

3. Exploratory Data Analysis (EDA)

The first part of the EDA was checking color consistency and image size. All the images in the datasets are color images with a size of 128x128x3. Once the images are checked for color consistency and size, features are extracted.

Two sets of features were extracted from the datasets:

- Color features --- the easiest and quickest feature to extract is the color feature
- Resnet50 features --- acquired features from a pre-trained model. Besides color features, these features provide more information about the shape, corners and much more complicated features

3.1. Principal Component Analysis (PCA)

After getting the features, PCA analysis was done on the features to see if we can get principal components that can clearly separate the damage and no-damage datasets. Figure 3 below show the result of the PCA analysis. Using color only features was unable to separate the two classes of the dataset. However, when the same PCA analysis is done using the resnet50 features, there seems to be some separation of the damaged and no-damage classes with some overlap boundary. 3D PCA was also explored if we can see more separation, however the results were similar.

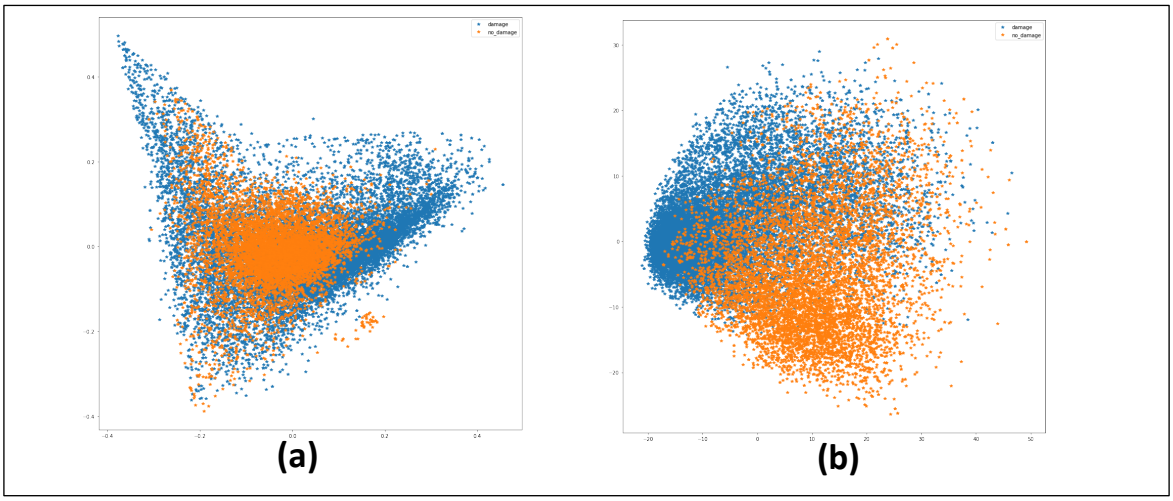


Figure 3 Principal component analysis (PCA). (a) PCA with color only features (b) using resnet50 features

4. Modeling

4.1. Computer vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information.

Computer vision is the broad parent name for any computations involving visual content – that means images, videos, icons, and anything else with pixels involved.[2]

Some of the core building blocks of computer vision include:

- **Object classification** -- you train a model on a dataset of specific objects, and the model classifies new objects as belonging to one or more of your training categories.

- **Object identification --** your model will recognize a specific instance of an object – for example, parsing two faces in an image and tagging.
- **Video motion analysis** -- to estimate the velocity of objects in a video, or the camera itself.
- **Image segmentation** -- algorithms partition images into multiple sets of views.
- **Scene reconstruction** -- creates a 3D model of a scene inputted through images or video.
- **Image restoration** -- noise such as blurring is removed from photos using Machine Learning based filters.

Much of the progress made in computer vision accuracy over the past few years is due in part to a special type of algorithm called Convolutional Neural Networks (CNN). CNNs are a subset of Deep Learning with a few extra added operations, and they've been shown to achieve impressive accuracy on image-associated tasks.

CNNs utilize the same major concepts of Neural Networks, but add in some steps before the normal architecture. These steps are focused on feature extraction, or finding the best version possible of our input that will yield the greatest level of understanding for our model. Ideally, these features will be less redundant and more informative than the original input. We will be utilizing CNN to build our models.

4.2. Transfer learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.[3]

The following is the approach taken to do transfer learning for this project

1. **Select Source Model**. A pre-trained source model (in this case VGG16) is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
2. **Reuse Model**. The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. **Tune Model**. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

4.3. VGG16 model

VGG16 is a simple and widely used Convolutional Neural Network (CNN) Architecture used for ImageNet, a large visual database project used in visual object recognition software research. The VGG16 architecture was developed and introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford, in the year 2014. The VGG16 model achieved 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. VGG16 Architecture has 16 layers as can be seen in the figure 4 [4] below.

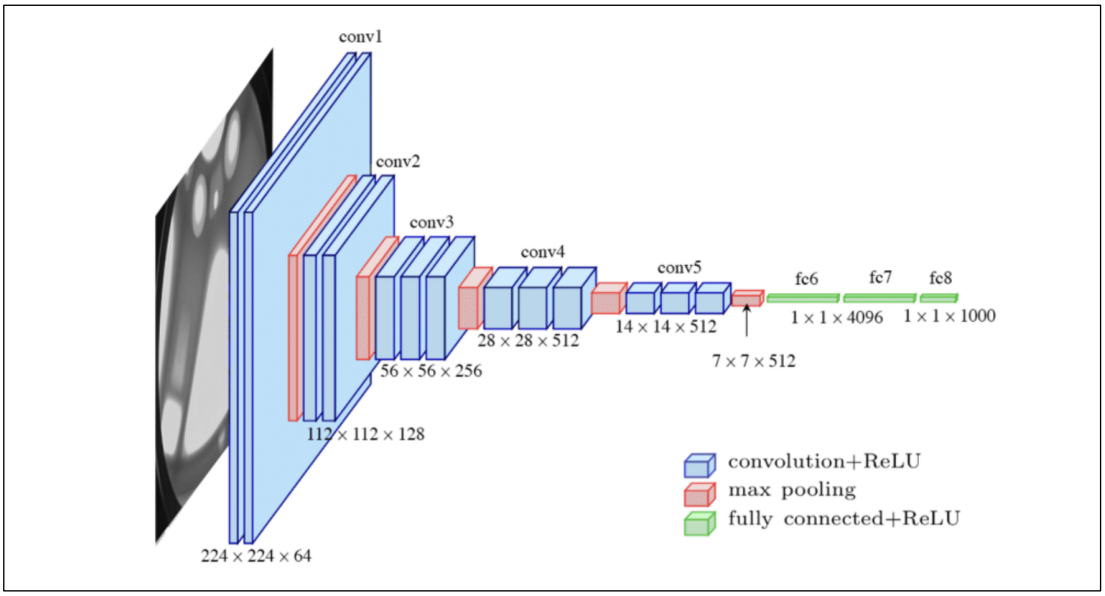


Figure 4 VGG16 architecture.

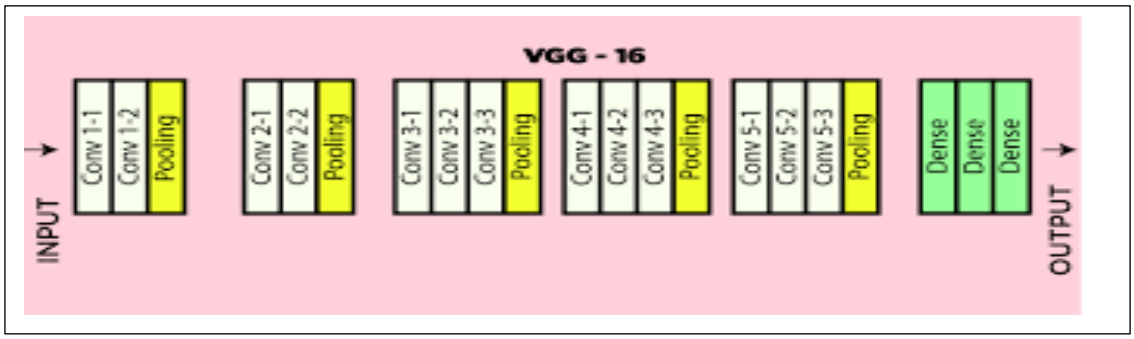


Figure 5 VGG16 layers

4.4. Performance Metrics

The performance of each model was measured using different metrics including accuracy, precision, recall, confusion matrix and classification report. In addition, ROC and AUC values will be examined to access the quality of the classification from the models.

4.5. Base Line Model

Before we embark on building CNN models to classify our images, we need a baseline model to be used as a benchmark. We expect our final model to be better than the baseline model.

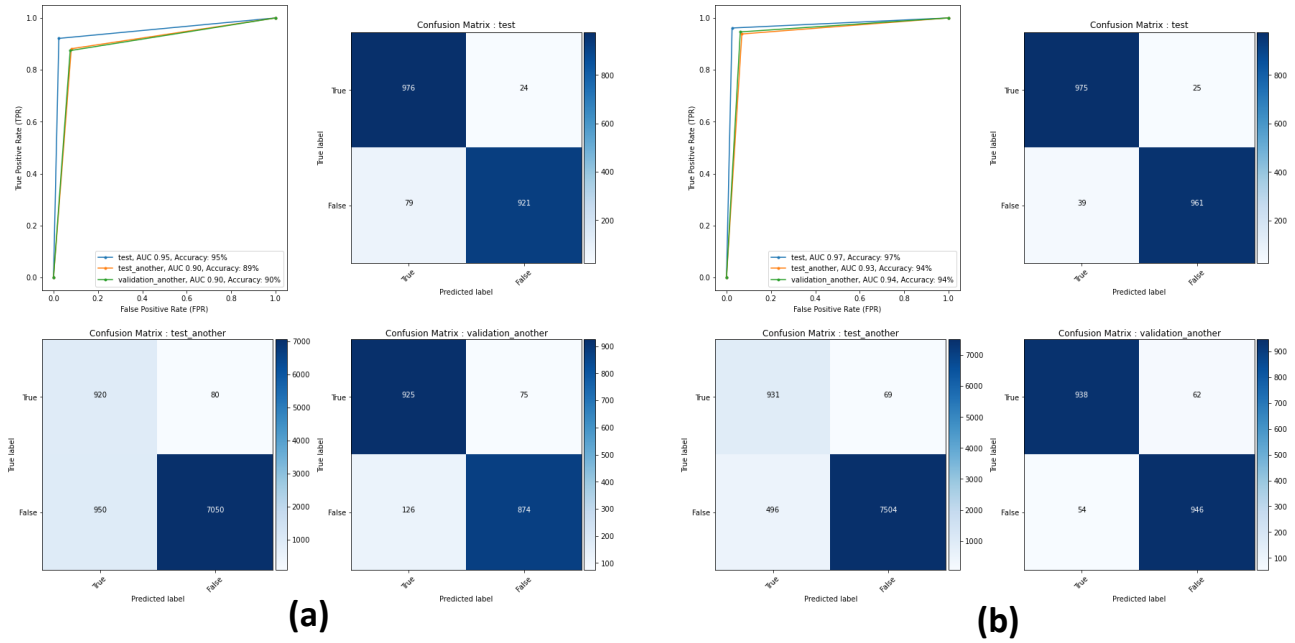
We build two simple baseline models using K-Nearest Neighbors (KNN) algorithm. Two kinds of features were utilized to build the two baseline models:

- Baseline model 1: KNN model using color features only
- Baseline model 2: KNN model using features extracted from resnet50 pre-trained model. These include many features beside the color features

The baseline model 1 built using the color features only actually performed reasonably well (figure 6a). AUC of 90% for test_another & validation datasets and 95% for the test data were achieved. Similarly, the accuracy of baseline model 1 are 89%, 90% and 95% for the test_another (unbalanced datasets), validation and test datasets respectively. However, when the test dataset is highly

unbalanced, as in the case of the test_another (1000 no-damage class vs 8000 damage class), the precision of the simple baseline model suffers a lot. It produced a precision of 49% on the unbalanced data compared to ~ 90% precision for the balanced datasets.

In comparison, baseline model 2, which uses features extracted from resnet50 model, produced an increase of ~ 4% in AUC and accuracy for all the datasets ((figure 6b). Moreover, the precision of the unbalanced data increased from 45% (in the baseline model1) to 65%.



Figures 6 AUC curve and confusion matrix. (a) Baseline Model 1, (b) Baseline model 2

Overall, the performance of these baseline models is surprisingly good. For a simple and fast model with minimum effort, we were able to achieve an accuracy of ~94%. The classification report of the baseline models is summarized in the figure 7. In the next section we start building our model using transfer learning from VGG16.

		Base Line Model - Color Features				Base Line Model - Resnet Features			
Dataset		precision	recall	f1-score	support	precision	recall	f1-score	support
Test	FALSE	0.925	0.976	0.950	1000	0.962	0.975	0.968	1000
	TRUE	0.975	0.921	0.947	1000	0.975	0.961	0.968	1000
	accuracy	0.949	0.949	0.949	0.949	0.968	0.968	0.968	0.968
	macro avg	0.950	0.949	0.948	2000	0.968	0.968	0.968	2000
	weighted avg	0.950	0.949	0.948	2000	0.968	0.968	0.968	2000
Test Another	FALSE	0.492	0.920	0.641	1000	0.652	0.931	0.767	1000
	TRUE	0.989	0.881	0.932	8000	0.991	0.938	0.964	8000
	accuracy	0.886	0.886	0.886	0.886	0.937	0.937	0.937	0.937
	macro avg	0.740	0.901	0.787	9000	0.822	0.935	0.865	9000
	weighted avg	0.934	0.886	0.900	9000	0.953	0.937	0.942	9000
Validation Another	FALSE	0.880	0.925	0.902	1000	0.946	0.938	0.942	1000
	TRUE	0.921	0.874	0.897	1000	0.938	0.946	0.942	1000
	accuracy	0.900	0.900	0.900	0.900	0.942	0.942	0.942	0.942
	macro avg	0.901	0.900	0.899	2000	0.942	0.942	0.942	2000
	weighted avg	0.901	0.900	0.899	2000	0.942	0.942	0.942	2000

Figures 7 Classification report of the base line models

4.6. Extended Modeling

As mentioned above, the plan for this project is to build CNN model by using transfer learning from VGG16. Two models were generated:

1. Model 1
   - Take all the convolutional + the max pooling layers of VGG16 and freeze them
   - Add a global-average layer and a prediction layer

2. Model 2
   - Take all the convolutional + the max pooling layers of VGG16 and freeze the top 15 layers only. The deeper three convolutional layers would be retrained using our input datasets
   - Add a global-average layer and a prediction layer

4.6.1. Model 1

As can be seen from the figure 8, Model 1 adopts all the convolutional and max pooling layers of the VGG16. This created a total of 14714688 parameters with

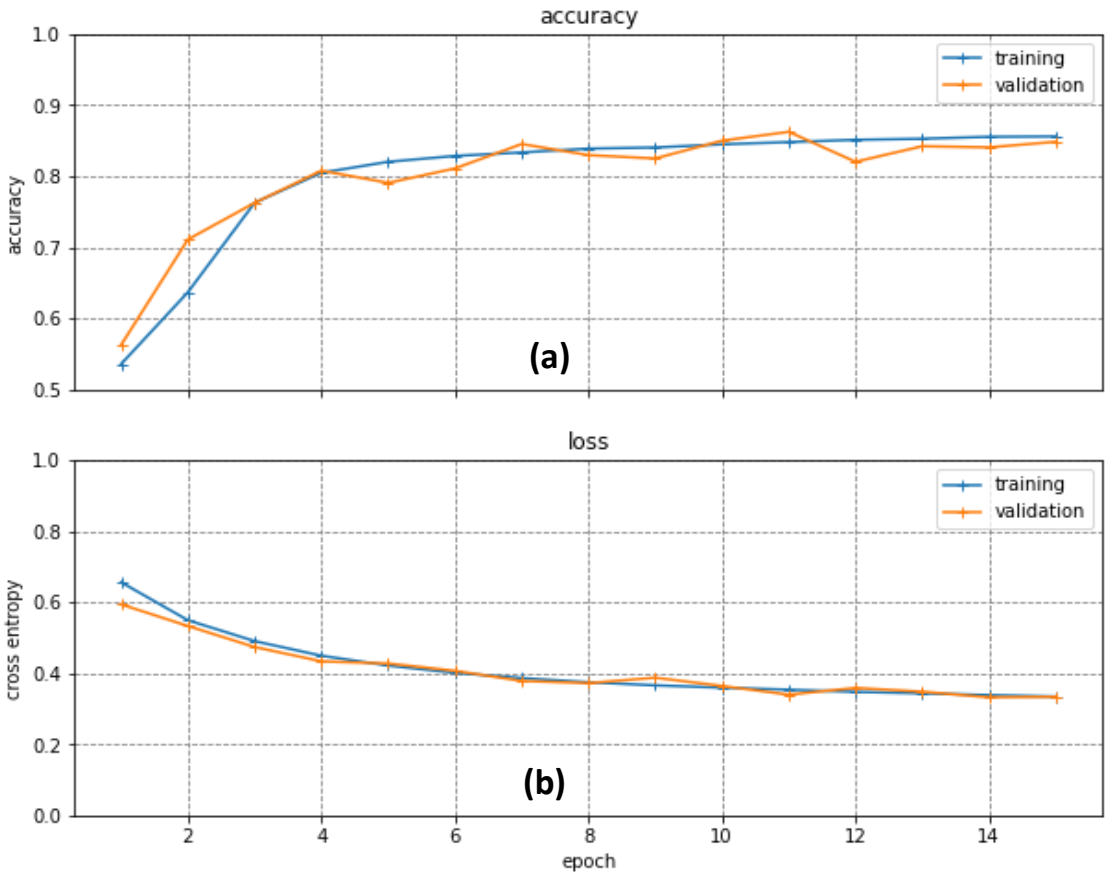
the final layer output of 4x4x512. Since we have frozen all this layers, none of the 14.7M parameters are trainable. Only 513 parameters are trainable. In total model 1 architecture has 14715201 parameters.

```
Model - 1 summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 513
=================================================================
Total params: 14,715,201
Trainable params: 513
Non-trainable params: 14,714,688
```

Figures 8 Summary of Model 1

With the above architecture, model 1 was compiled using Adam optimizer with a learning rate of 0.0001 and it was trained for 15 epochs with a batch size of 32.

Figure 9 shows the loss and accuracy of each epoch. The accuracy of the model improved sharply in the first 4 iterations (increased from54% to 80%) and then in the remaining 11 iteration the improvement was very small. The plot of the accuracy flattens after the fourth iteration. Similarly, the loss value dropped sharply in the first 4 iteration and then flattens out.



Figures 9 Accuracy and loss value of each epoch in model 1. (a) Accuracy and (b) Loss value

Model 1 achieved an accuracy of only 86% on the training, validation and test datasets. The result shows that the model has generalized very well and no issue of over-fitting. However, the accuracy is way lower than both the base models we created based on KNN using color features as well as features generated from RESNET50. We need to build a model that can achieve better accuracy than the base model.

4.6.2. Model 2

Model 2's architecture is similar to model 1 but model 2 re-trains the last three convolutional layers of the VGG16 architecture. This means we only freeze the top 15 layers and re-train all the deeper layers with our datasets.

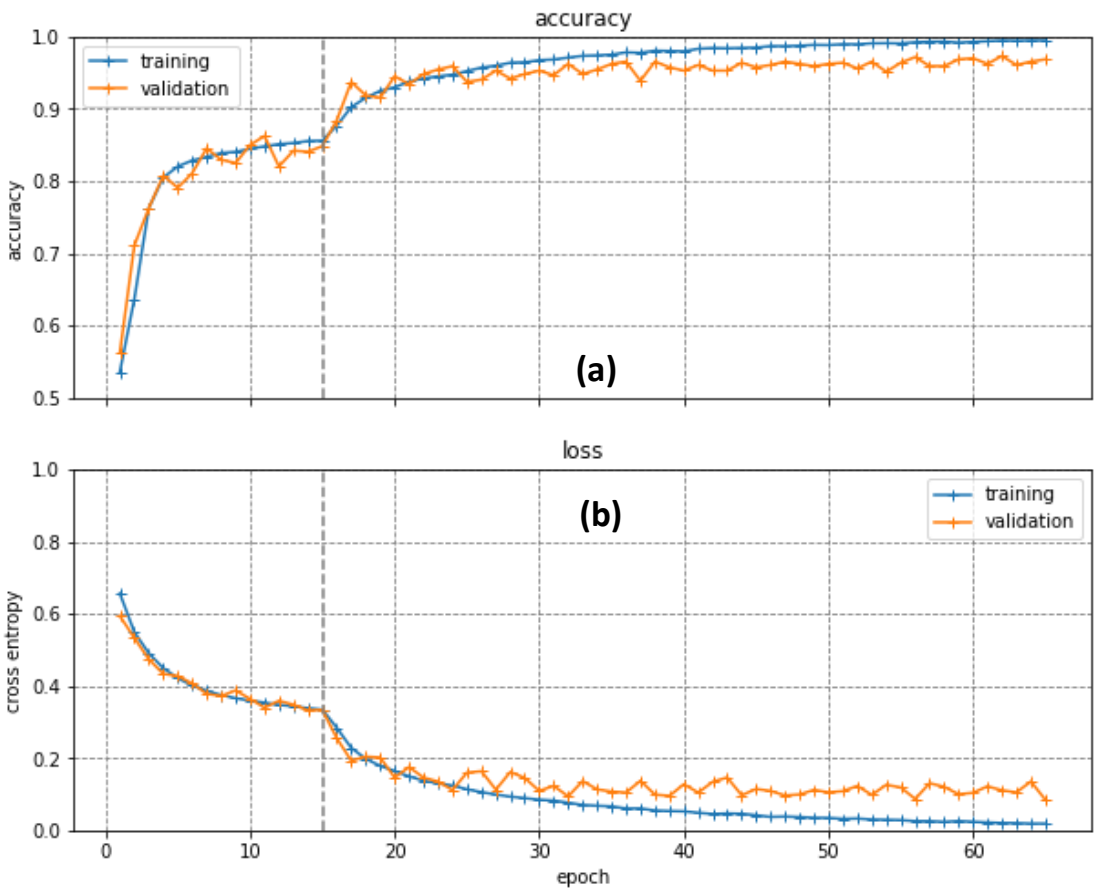
```
Model - 2 summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 513
=================================================================
Total params: 14,715,201
Trainable params: 7,079,937
Non-trainable params: 7,635,264
```

Figures 10 Summary of Model 2

The total number of parameters are the same as model 1 but the number trainbale parameters are way higher than model 1 (see figure 10). Model 2 has 7,079,937 parameters to train, in comparison model 1 has only 513 parameters to train. This will make training model 2 to be more expensive and time consuming

Model 2 was also compiled using Adam optimizer with 50 epochs to fine tune the parameters. The 50 epochs are run on top of the model 1 results, making a total of 65 epochs.

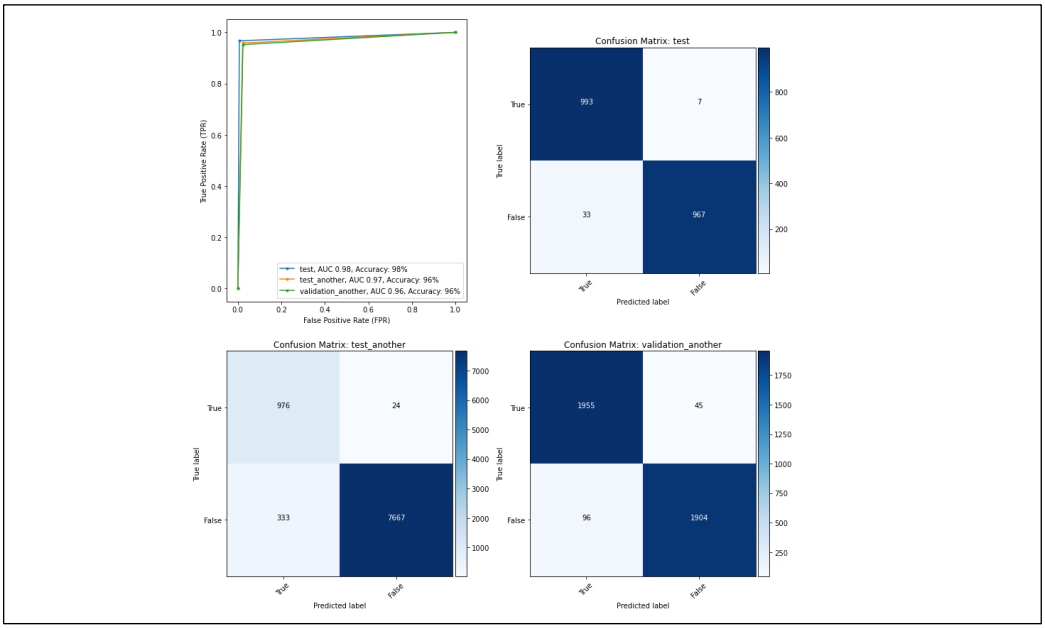
Figure 11 below show the accuracy and the losses of each epoch. There is a sharp increase in accuracy at iteration 16 (where the fine-tunning of model 2 starts) and then it steadily increases as the number of epcohs goes up. Likewise, the loss value drops sharply at iteration 16 and lowers steadily until it flattens out.



Figures 11 Accuracy and loss value of each epoch in model 2. (a) Accuracy and (b) Loss value

Model 2 results are much better compared to model 1 and the base line models (see figure 12). It has achieved an accuracy of 96% for the unbalanced test dataset. Accuracy of 86% and 93% were achieved by model 1 and the baseline model using resnet50 respectively. The AUC values are 98%, 97% and 96% for the test data, the unbalance test data and the validation data respectively. In comparison the AUC number from the baseline model were 97, 93% and 94% respectively.

Fine tuning the parameters of the deeper few convolutional layers has improved the accuracy by almost 10%. This suggests that the deeper layers, which create features that are specific to the dataset, need to get trained on the dataset at hand.



Figures 12 AUC curve and confusion matrix of model 2

5. Findings

We used VGG16 model to build our finest and best model. We built two models out of it. One (Model 1) that fixes all the convolutional layers and another (Model 2) that re-trains the last three convolutional layers. The best results were achieved by model 2 and hence will be used as our final model.

Model 2 has out-performed the baseline model significantly, especially when the dataset was unbalanced. The table below summarized the results of the baseline and final model.

		Base Line Model - Color Features				Base Line Model - Resnet Features				Final Model			
Dataset		precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support
Test	FALSE	0.925	0.976	0.950	1000	0.962	0.975	0.968	1000	0.968	0.993	0.980	1000
	TRUE	0.975	0.921	0.947	1000	0.975	0.961	0.968	1000	0.993	0.967	0.980	1000
	accuracy	0.949	0.949	0.949	0.949	0.968	0.968	0.968	0.968	0.980	0.980	0.980	0.980
	macro avg	0.950	0.949	0.948	2000	0.968	0.968	0.968	2000	0.980	0.980	0.980	2000
	weighted avg	0.950	0.949	0.948	2000	0.968	0.968	0.968	2000	0.980	0.980	0.980	2000
Test Another	FALSE	0.492	0.920	0.641	1000	0.652	0.931	0.767	1000	0.746	0.976	0.845	1000
	TRUE	0.989	0.881	0.932	8000	0.991	0.938	0.964	8000	0.997	0.958	0.977	8000
	accuracy	0.886	0.886	0.886	0.886	0.937	0.937	0.937	0.937	0.960	0.960	0.960	0.960
	macro avg	0.740	0.901	0.787	9000	0.822	0.935	0.865	9000	0.871	0.967	0.911	9000
	weighted avg	0.934	0.886	0.900	9000	0.953	0.937	0.942	9000	0.969	0.960	0.963	9000
Validation Another	FALSE	0.880	0.925	0.902	1000	0.946	0.938	0.942	1000	0.953	0.978	0.965	2000
	TRUE	0.921	0.874	0.897	1000	0.938	0.946	0.942	1000	0.977	0.952	0.964	2000
	accuracy	0.900	0.900	0.900	0.900	0.942	0.942	0.942	0.942	0.965	0.965	0.965	0.965
	macro avg	0.901	0.900	0.899	2000	0.942	0.942	0.942	2000	0.965	0.965	0.965	4000
	weighted avg	0.901	0.900	0.899	2000	0.942	0.942	0.942	2000	0.965	0.965	0.965	4000

Figures 13 Classification report of the base line models and the final model (model 2)

Even though our model's performance was very good, there are some images that we mis-classified. we investigate some of the mis-classified images. The figure 14 shows some images of the false positives and the false negatives.

The first two rows show images that has been falsely classified as damaged. Looking at the images visually, some of them do look flooded, or at least the images does not look clear. The presence of debris could also lead the model to mis-classification. The same observation can also be said for the images that are mis-classified as not-damaged. Looking at some of the images closely, the images don't look flooded at all. Maybe they are mis-labeled, especially if the background has some dirt road or small vegetations, it could mis-lead to classify them as flooded if not looked closely.

Figures 14 Sample Images of the mis-classified images. The first two rows are false positives and the last two are rows are false negatives

Finally, we compare our best results to the study done by Quoc Dung Cao and Youngjun Choe (original study associated with the data [5]) on this dataset. According to the paper published by the afore mentioned authors, the best results were obtained by a model built from scratch as CNN + data augmentation + 50% dropout using Adam optimizer. Their model has achieved an accuracy of 98%, 97.29% and 97.03% for the validation, test data(balanced) and another test (unbalanced data). In comparison our model 2 has achieved an accuracy of 96%, 98% and 96% for the validation, test data(balanced) and another test (unbalanced data).

From this comparison we can see that by doing transfer learning we can achieve similar performance to the ones built from scratch. This could save a lot of time and compute expenses.

It is good to mention that the problem at hand was only a binary classification and the images are sampled from a very localized geographic region, that could have similar features. This may have made the classification a little simpler.

6. Conclusion

We demonstrated that transfer learning + fine tuning of the deeper layers can achieve similar results to the CNN models built from scratch. This will save a lot of compute power and more importantly time. For emergency managers and responders to build situational awareness and response to hurricane, a quick and accurate identification of damage assessment is beneficial.

In fact, this can be done into two stages:

- Stage1 --- Extract features from a pre-trained model such as resnet50 or VGG16 and use the features as an exogenous parameter and run a quick classification model such as KNN or logistic linear regression. Though the accuracy is not at the level of the CNN models, it is very quick with decent accuracy.
- Stage2 --- While the emergency managers are utilizing the results from stage1, a new model can be trained using transfer learning + fine-tuning which would greatly improve the accuracy with minimum runtime.

7. Future Works

The baseline model generated based on features extracted from resnet50 produced really good results. This could be either:

- The dataset used were simple. The dataset came from a small geographic location and may have similar building
- The features extract from restnet50 are really good and were able to generalize over many images.

It would be good to do the same exercise using VGG16. Extract features from VGG16 and use KNN to build a model and compare results to that of the resnet50.

The comparison will confirm whether the good results of the baseline model were due to the features of the resnet50 or similar accuracy can be achieved by utilizing another pre-trained model.

Another test that could be done is to do transfer learning using the resnet50 architecture and compare it to our best model, which was generated by doing transfer learning from VGG16.

8. Recommendation

Instead of building CNN from scratch, it is good to consider and explore transfer learning first. There are several models easily available that were already trained on a very large dataset (ImageNet).

We can use these models in several ways:

- for a quick and decent results, we can just simply extract the features from pre-trained models and use other machine learning algorithms to do the classification or prediction. This could give a decent accuracy
- Another way to use transfer learning is, to re-train or fine-tune the last few deeper layer with the current datasets. This will allow the model to learn some features that are specific to the current data and hence improve the performance.

9. Resources

[1] https://www.kaggle.com/kmader/satellite-images-of-hurricane-damage

[2] https://www.datarobot.com/blog/introduction-to-computer-vision-what-it-is-and-how-it-works/

[3] https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[4] https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484

[5] https://arxiv.org/abs/1807.01688