Automating Detection of Flood Damaged Buildings

Anne Evered
Final Project
MUSA650 Spring 2020

Background and Data Overview

Assessing Flood-Damaged Buildings in the Context of Emergency Response

Background

Assessment of damaged buildings is a critical step in emergency response after hurricanes and other severe weather events.

Such assessments help **guide where response is most needed**, as well as **provide an overall picture of infrastructure damage** in a region.

However, evaluating which buildings have sustained flood damage during hurricane events can be a **difficult and time intensive process**.

Remote Sensing (RS) and Geographic Information Technologies (GIS) technologies provide opportunities for automating parts of this detection process.

Project Goal: Development of a model that given an image, can detect if the building in that image likely has flood damage.

Data Overview and Preparation

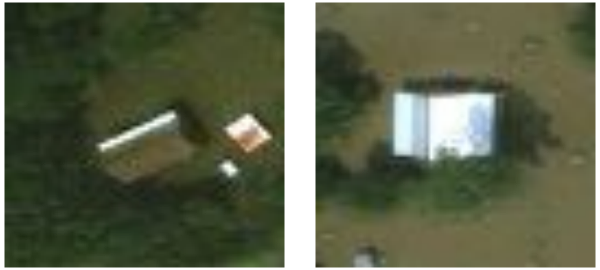
23,000 satellite images from Houston, TX after Hurricane Harvey hit the region in 2017

Labeled as **'Flooded/Damaged' or 'Undamaged'** based on building condition

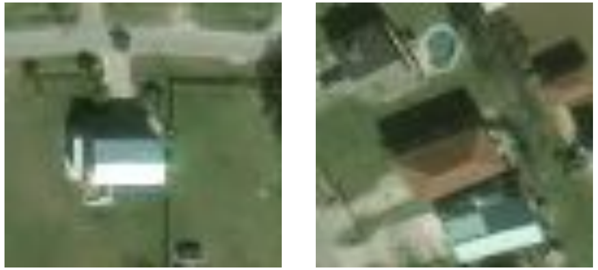
Image Size: **128x128x3**

Pre-segmented into validation, testing and training sets

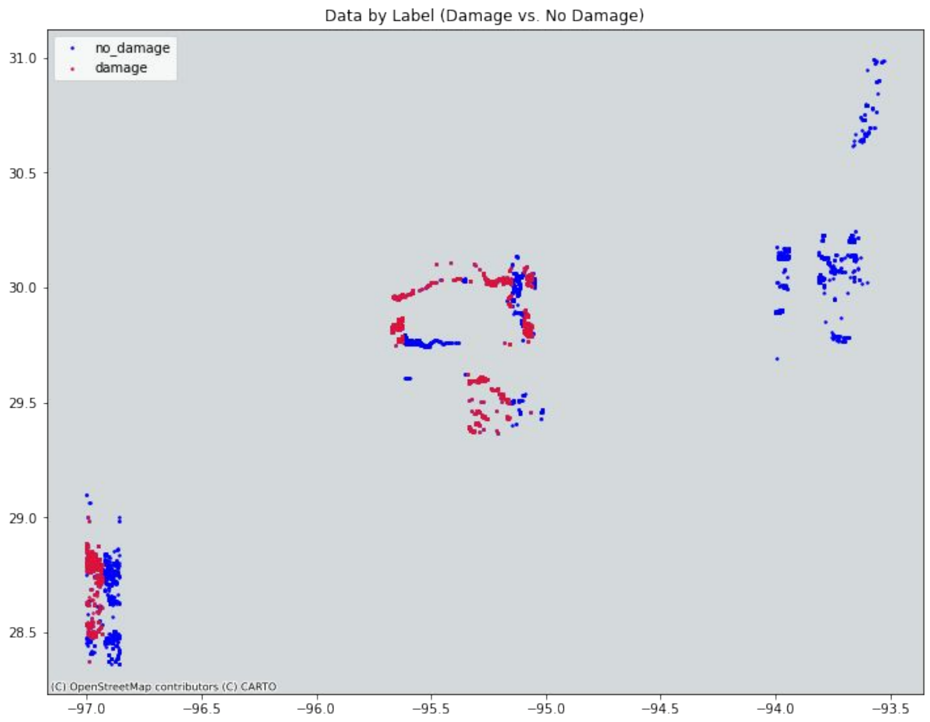
Divided each value by 255 to **standardize**



Label = “damage”



Label = “no damage”

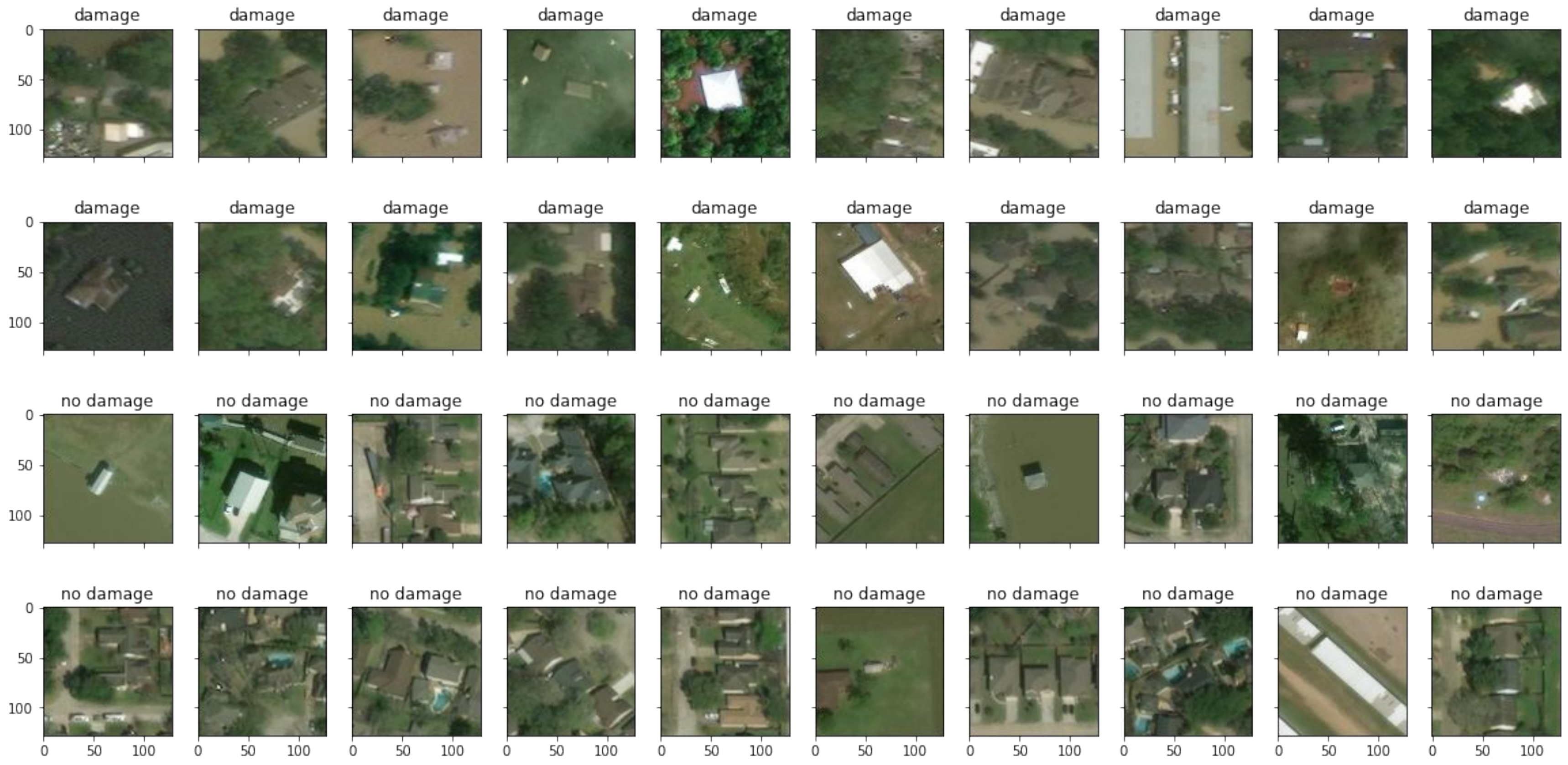


Coordinates of Satellite Images

Test/Train/Validation Set	Label	Number of Images
test	damage	1,000
	no_damage	1,000
test_another	damage	8,000
	no_damage	1,000
train_another	damage	5,000
	no_damage	5,000
validation_another	damage	1,000
	no_damage	1,000

Number of Images per Data Segmentation and Label Category

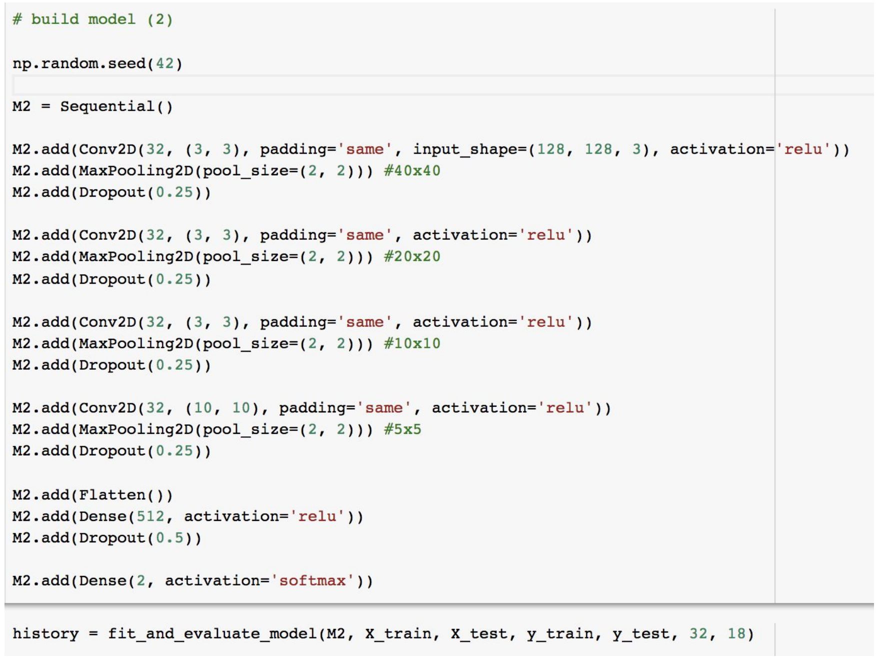
(1) Distribution of Images Geographically and (2) Number of Images per Category



Examples of images

Methods

Model Development and Analysis



```
# build model (2)

np.random.seed(42)

M2 = Sequential()

M2.add(Conv2D(32, (3, 3), padding='same', input_shape=(128, 128, 3), activation='relu'))
M2.add(MaxPooling2D(pool_size=(2, 2))) #40x40
M2.add(Dropout(0.25))

M2.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
M2.add(MaxPooling2D(pool_size=(2, 2))) #20x20
M2.add(Dropout(0.25))

M2.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
M2.add(MaxPooling2D(pool_size=(2, 2))) #10x10
M2.add(Dropout(0.25))

M2.add(Conv2D(32, (10, 10), padding='same', activation='relu'))
M2.add(MaxPooling2D(pool_size=(2, 2))) #5x5
M2.add(Dropout(0.25))

M2.add(Flatten())
M2.add(Dense(512, activation='relu'))
M2.add(Dropout(0.5))

M2.add(Dense(2, activation='softmax'))
```

```
history = fit_and_evaluate_model(M2, X_train, X_test, y_train, y_test, 32, 18)
```

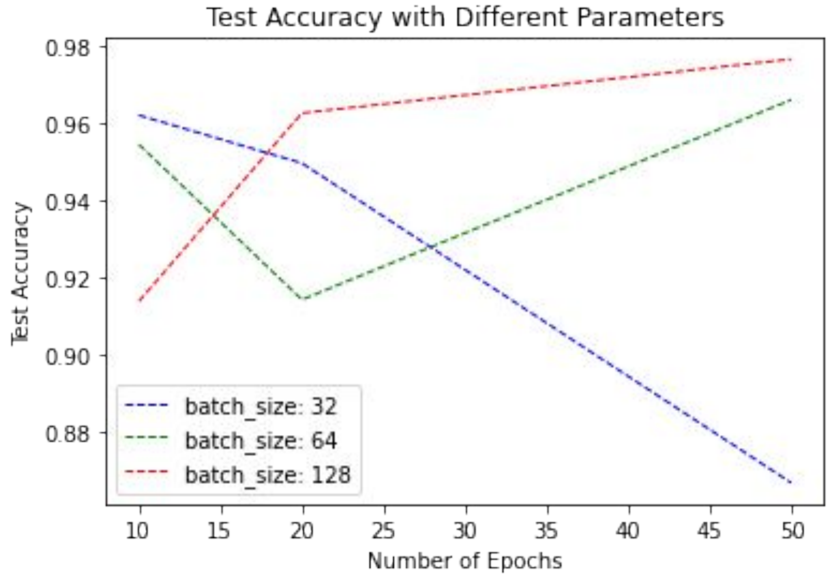
Model Development

Tested series of **Convolutional Neural Networks,** ranging from single dense layer with gray-scale images to multilayer CNNs with multiple drop out and pooling layers

Adjusted Drop-out and Pooling Layers, Number of Epochs, Batch Sizes

For all models used **Categorical Cross-entropy loss function** and **Softmax activation**

After testing different models (with batch size: 32, number of epochs: 50), I **chose the model with best accuracy and determined best epoch number and batch size**



Results of parameter testing for model structure with best performance

Final Model

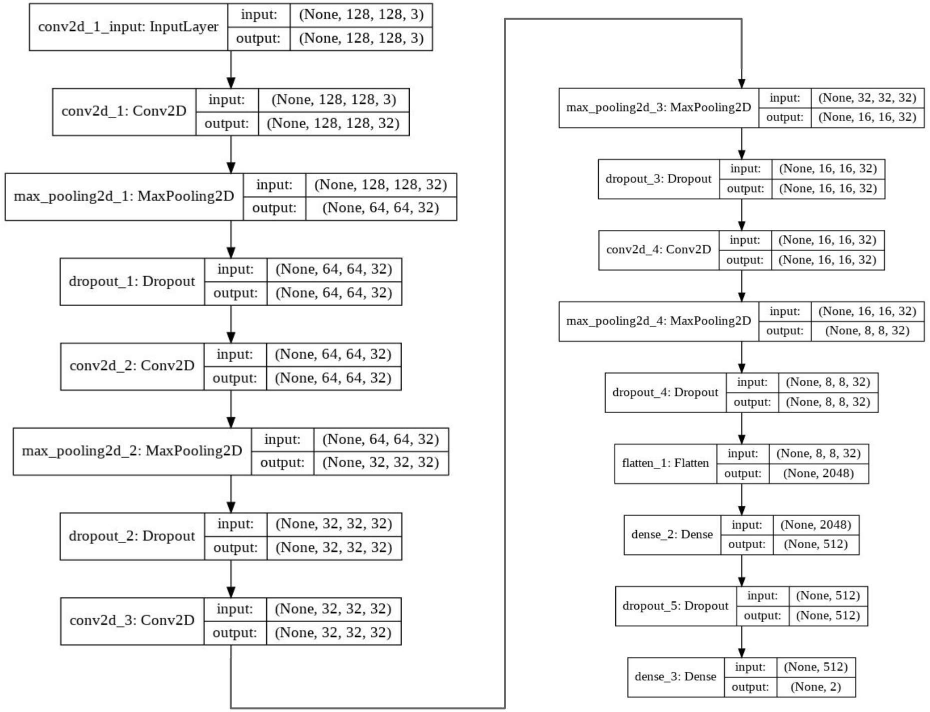
Multi-layer CNN Network (summary at right)

Batch Size: 128

Number of Epochs: 50

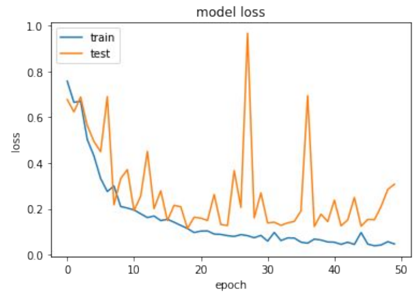
Loss Function: Categorical Cross-entropy loss

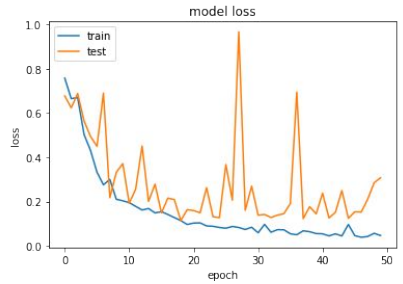
Activation: Softmax

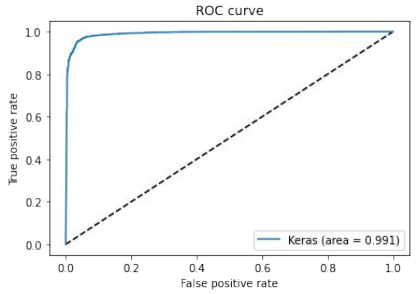

conv2d_1_input: InputLayer | input: (None, 128, 128, 3) | output: (None, 128, 128, 3)
conv2d_1: Conv2D | input: (None, 128, 128, 3) | output: (None, 128, 128, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 128, 128, 32) | output: (None, 64, 64, 32)
dropout_1: Dropout | input: (None, 64, 64, 32) | output: (None, 64, 64, 32)
conv2d_2: Conv2D | input: (None, 64, 64, 32) | output: (None, 64, 64, 32)
max_pooling2d_2: MaxPooling2D | input: (None, 64, 64, 32) | output: (None, 32, 32, 32)
dropout_2: Dropout | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
conv2d_3: Conv2D | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
max_pooling2d_3: MaxPooling2D | input: (None, 32, 32, 32) | output: (None, 16, 16, 32)
dropout_3: Dropout | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
conv2d_4: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
max_pooling2d_4: MaxPooling2D | input: (None, 16, 16, 32) | output: (None, 8, 8, 32)
dropout_4: Dropout | input: (None, 8, 8, 32) | output: (None, 8, 8, 32)
flatten_1: Flatten | input: (None, 8, 8, 32) | output: (None, 2048)
dense_2: Dense | input: (None, 2048) | output: (None, 512)
dropout_5: Dropout | input: (None, 512) | output: (None, 512)
dense_3: Dense | input: (None, 512) | output: (None, 2)

Results and Evaluation

Model Performance and Considerations



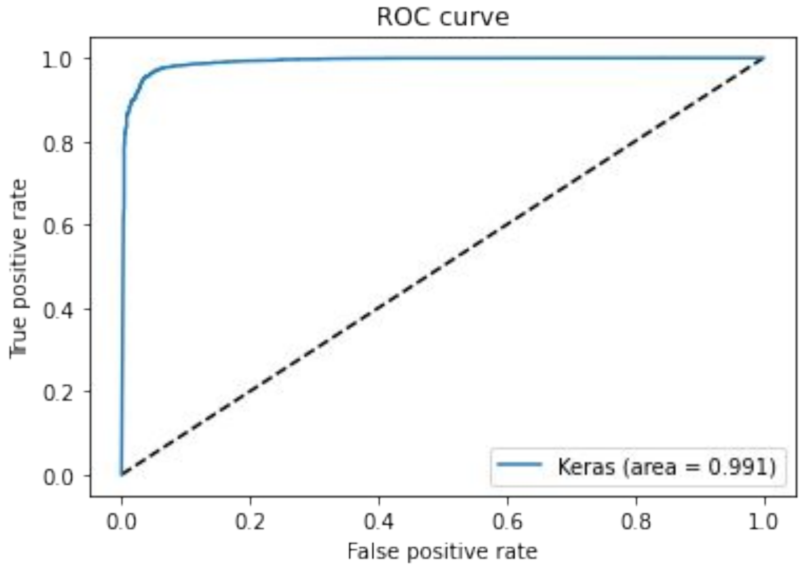




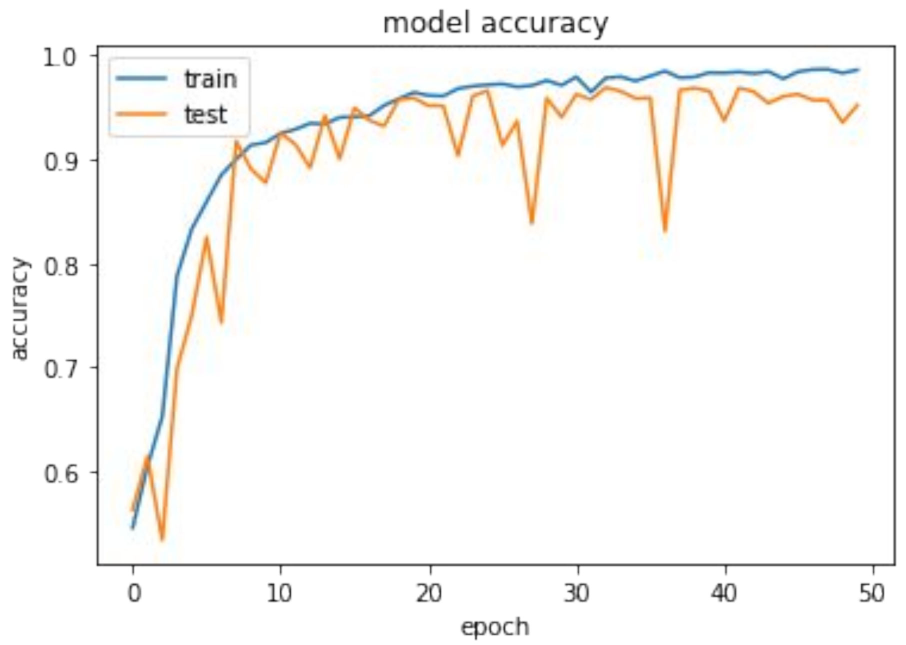
Model Performance (for best model)

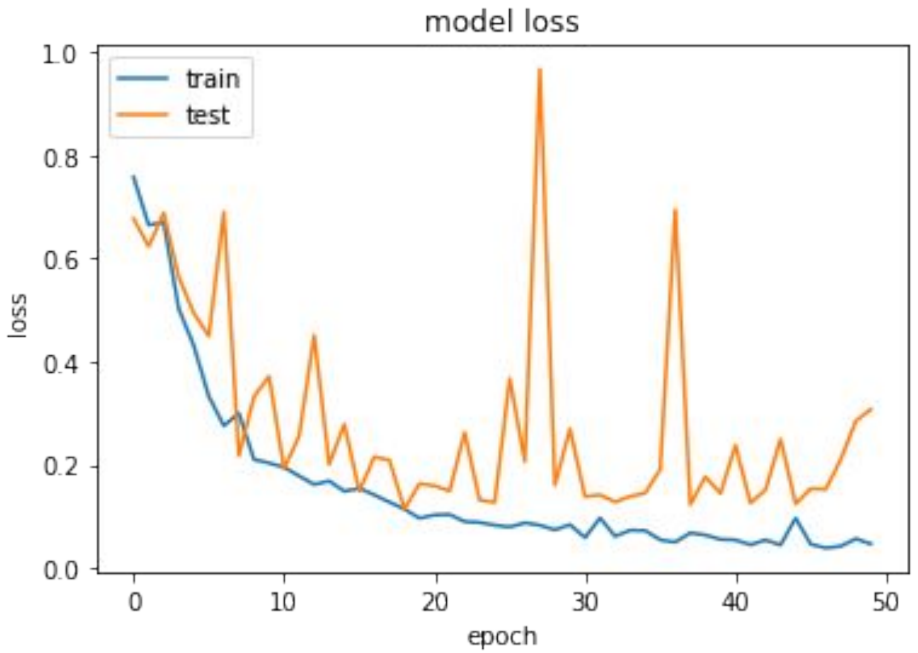
Accuracy Score: 96.877%
Precision Score: 99.272%
Recall Score: 97.200%
F1 Score: 98.225%

True Label	Correct Predictions	Total	Accuracy
damage	7776	8000	97.200%
no_damage	943	1000	94.300%



All of the above metrics (and the figures and misclassified images that follow) are from testing the model on the full unbalanced testing dataset. I also tested the model on the smaller balanced testing dataset and got a testing accuracy of 97.600%, suggesting that the model also performs well on the balanced testing data.

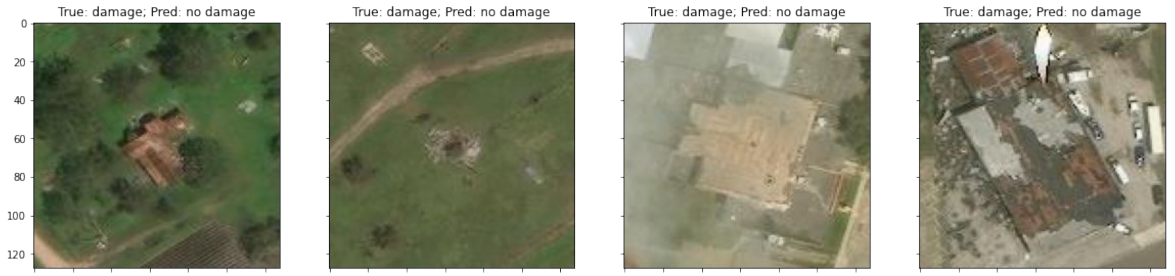




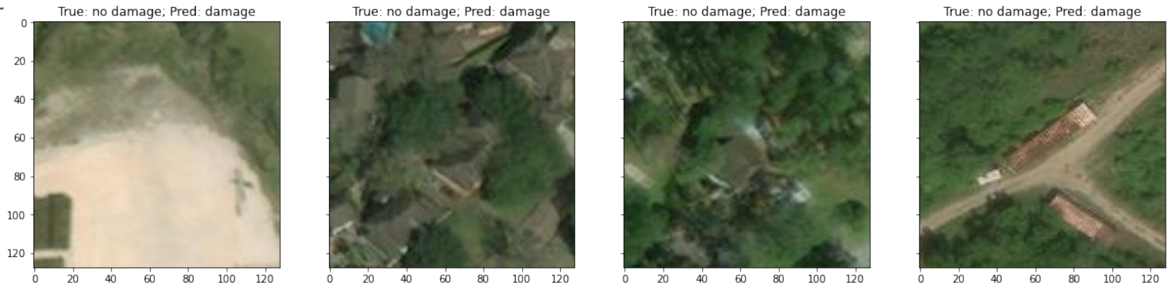
Model loss and accuracy by epoch for best accuracy model with unbalanced testing data

Misclassified Images Examples

Misclassified as not damaged
(True label: damage)


True: damage; Pred: no damage
True: damage; Pred: no damage
True: damage; Pred: no damage
True: damage; Pred: no damage

Misclassified as damaged
true label: no_damage


True: no damage; Pred: damage
True: no damage; Pred: damage
True: no damage; Pred: damage
True: no damage; Pred: damage

Discussion

Discussion of Results and Areas for Further Analysis

Discussion of Results

Like previous work, these results suggest that **convolutional neural networks show great promise at being able to differentiate between images that show post-hurricane flood damage and those that do not** with high accuracy. I was surprised at how well the CNNs were able to perform on this task given how subtle the differences seemed between the images from the two label categories.

However, a fairly complex CNN is required, with the fewer layer models not performing as well as the more complex one. In addition, the **RGB channels of the images appear important in achieving a high accuracy value**. This may be because the color channels help the model initially identify what is a building and what is landscaping.

One interesting result is that for all of the models that performed relatively well, the **validation accuracy is very high, even on the first epoch**. This may suggest a high redundancy in the data, with the model not learning much additional information in subsequent epochs.

Areas for Further Analysis

- Test the models on **satellite images from other regions and contexts**.
  - For example, see how the classification model performs on building images from Houston after a different storm or severe weather event or, for instance, from New York City after Hurricane Sandy. This may also help to identify which aspects of the images the model is most relying on for classification.
- **Apply additional deep learning techniques**, such as data augmentation and transfer learning, or try additional adjustments to network layers, to try further increasing test accuracy.
- Additional **research into the speed and accuracy of pre-processing work needed to run the CNN models** (i.e. creating and labeling testing and training datasets) to help increase the impact of automated detection of post-disaster building damage.
  - Particularly given that damage assessment and response after severe weather events is often time sensitive, in order for the application of machine learning to have a positive impact in this realm, an efficient preprocessing data pipeline is critical.

References

Dataset Citation:

Quoc Dung Cao, Youngjun Choe, "Detecting Damaged Buildings on Post-Hurricane Satellite Imagery Based on Customized Convolutional Neural Networks", IEEE Dataport, 2018. [Online]. Available: http://dx.doi.org/10.21227/sdad-1e56. Accessed: May. 04, 2020.

Dataset also available for download on Kaggle:

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

Additional Papers Referenced:

Gua, Jiuxiang, et al., "Recent Advances in Convolutional Neural Networks." In: \emph{CoRR} https://arxiv.org/pdf/1512.07108.pdf (2017).

Park, Sung Jae et al., "Machine Learning Application for Coastal Area Change Detection in Gangwon Province, South Korea Using High-Resolution Satellite Imagery." In: \emph{J. of Coastal Research} pp. 228-235. https://bioone.org/journals/Journal-of-Coastal-Research/volume-90/issue-sp1/SI90-028.1/Machine-Learning-Application-for-Coastal-Area-Change-Detection-in-Gangwon/10.2112/SI90-028.1.short (2019).

Image Links

https://www.houstonchronicle.com/business/article/Flooded-by-Harvey-BP-s-main-Houston-office-tower-12199318.php
https://communityimpact.com/houston/conroe-montgomery/city-county/2017/09/23/conroe-homes-businesses-rebuild-hurricane-harvey/
https://www.theverge.com/2017/9/3/16250146/houston-superfund-sites-flooded-after-hurricane-harvey-epa