Machine Learning II Final Project Paper- Group 4

Flood Damage Detection Using Convolution Neural Network

Abdulaziz Gebril

April 27, 2021

Contents

Abstract

The damage assessment process post a Hurricane is imperative for the efficiency of relief attempts. In this project, automating the process of identifying damage buildings through satellite imagers have been proposed. Different convolutional neural networks were trained on data obtained from Kaggle that consisted of satellite images of Greater Huston area a post Hurricane Harvey in 2017. The models were then tested on balanced and unbalanced datasets achieving an overall accuracy higher than 90%. Details of how the models were trained and evaluated are presented in this report.

Introduction

After a hurricane, damage assessment is critical to emergency managers and first responders so that resources can be planned and allocated appropriately. One way to gauge the damage extent is to detect and quantify the number of damaged buildings, which is traditionally done through driving around the affected area. This process can be labor intensive and time-consuming. therefore, utilizing the availability and readiness of satellite imagery, the efficiency and accuracy of damage detection can be improved using image classification algorithms. In this project, Various convolutional neural network architectures were trained to classify images obtained from satellite images.

Description of the Dataset

The dataset was featured in a research paper by Quoc Dung Cao and Youngjun Choe published in the Institute of Electrical and Electronics Engineering Journal (December, 2018). The dataset consists of satellite imagery of the Greater Huston area before and after Hurricane Harvey in 2017. The flooded/damaged buildings were labeled by volunteers through crowd-sourcing project as shown in *Figure 1.* The green circles represent the areas that were labeled as flooded/damaged.

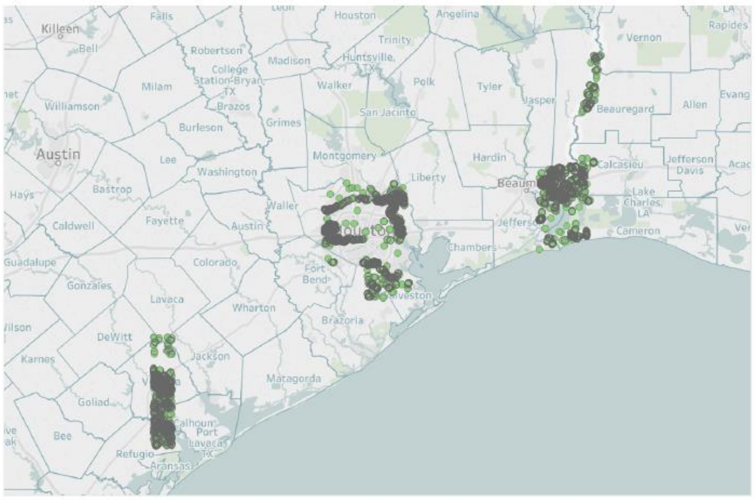


Figure 1. Greater Huston are affected by Hurricane Harvey 2017

The dataset consists of 23,000 3-Pixel RGB satellite images labeled into two classes, either “Damaged” or “No Damage”. The dataset is split into four sets as seen in *Table 1.*

Set	Damaged	No Damage
Training set	5,000	5,000
Validation	1,000	1,000
Unbalanced test set	8,000	1,000
Balanced test set	1,000	1,000

Table 1. Dataset split into training, validation, and test sets.

Pretrained models

There have been different methods responsible for the growth of deep learning, one of those is transfer learning. Transfer learning is a method that uses information learned by one trained model for another model that needs to be trained on different data and for a similar/different task. In this project, VGG-16, ResNet50 and AlexNet were chosen as pretrained models.

VGG-16

VGG-16 is a convolutional neural network model proposed by K.Simonyan and A.Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks For Large-Scale Image Recognition”. The model achieves 92.7% top-5 test accuracy in ImageNet dataset, which contains of over 14 million images belonging to 1000 classes. VGG-16 does not use large kernel-sized filters, instead uses 3*3 Convolutional layers from the beginning to the end of network as seen in *Figure 2*. This comes at a cost where it utilizes more memory and trainable parameters.

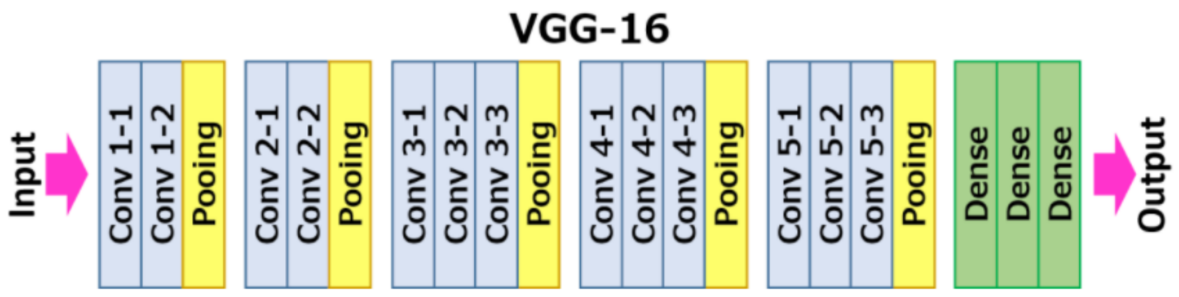


Figure 2. VGG-16 Model Architecture

ResNet50

ResNet is short for Residual networks , ResNet50 is similar to VGG-16 but with different kernel sizes and most importantly an additional block in its architecture which is the identity block as illustrated in *Figure 3.* ResNet models introduced the concept of skip connection. Skip connections migrate the problem of vanishing gradient by allowing alternate shortcut path for gradient to flow through as seen in *Figure 4*.

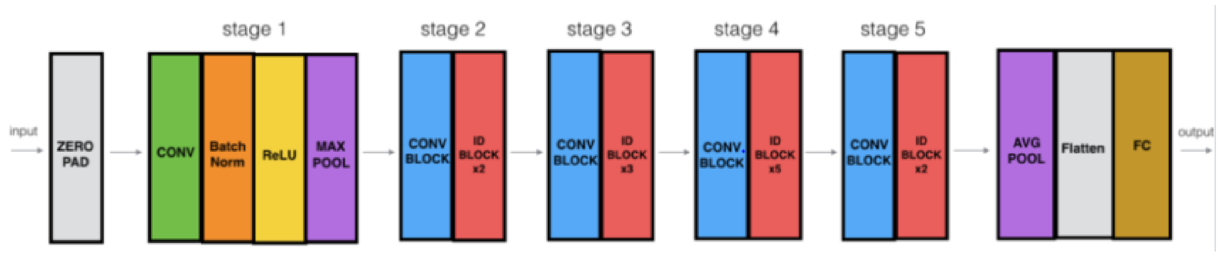


Figure 3. ResNet50 Model Architecture

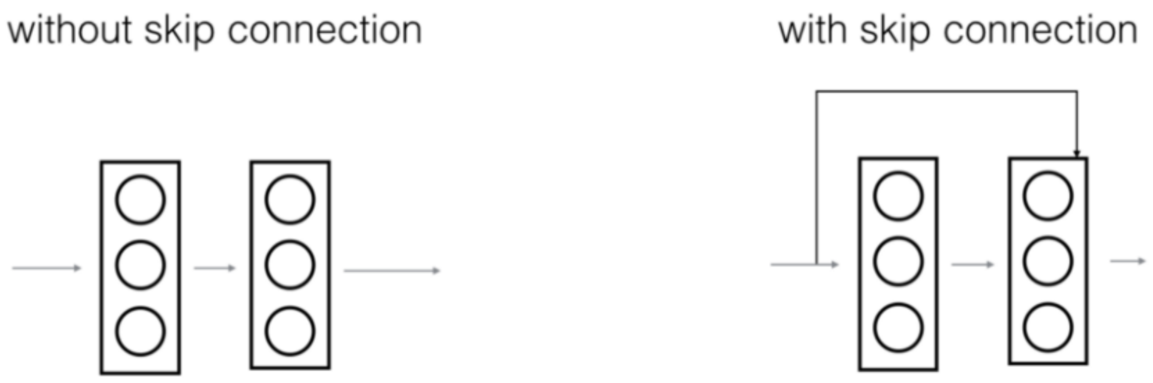


Figure 4. Skip Connections

AlexNet

AlexNet Consists of 5 convolutional layers and 3 fully connected layers. Unlike VGG-16, AlexNet uses large kernel sizes at the early convolutional layers as illustrated in *Figure 5*. AlexNet were the first to implement Rectified Linear Units as activation functions.

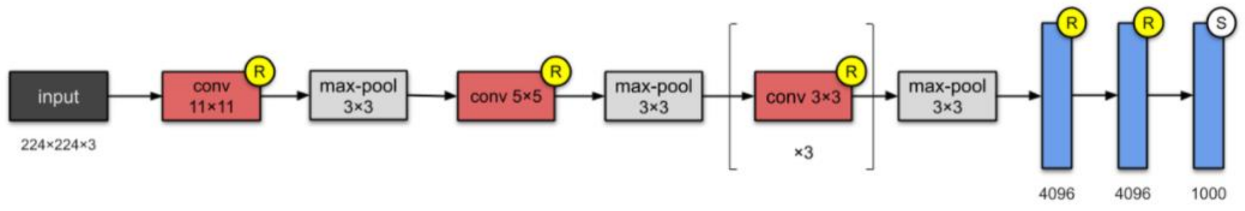
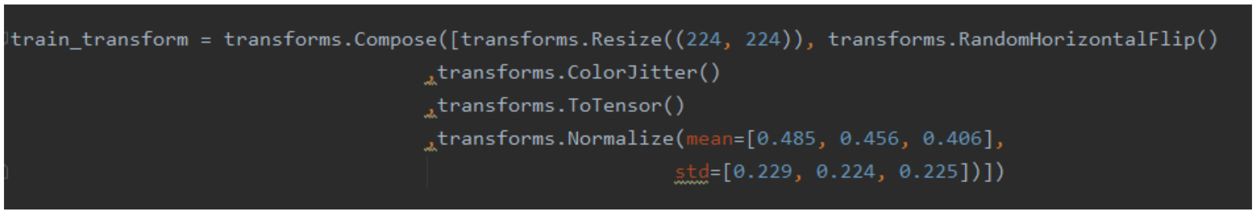


Figure 5. AlexNet Architecture

Pretrained model Experimental setup

Image Augmentation

Pretrained models requires images to be at a specific size (224,224) for VGG-19 and ResNet50 while a size (299,299) for AlexNet model. In addition, different augmentation techniques where performed such as normalizing all images and random horizontal flipping as seen in *Figure 6*.

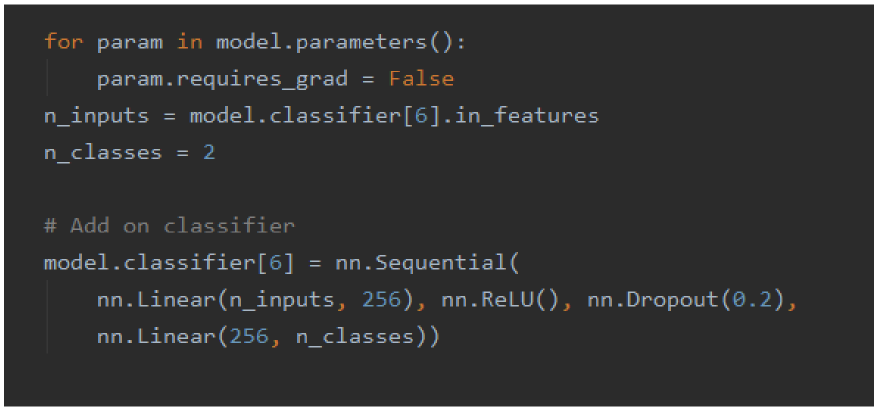


```
train_transform = transforms.Compose([transforms.Resize((224, 224)), transforms.RandomHorizontalFlip()
                                      ,transforms.ColorJitter()
                                      ,transforms.ToTensor()
                                      ,transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                            std=[0.229, 0.224, 0.225])])
```

Figure 6. Image Augmentation for AlexNet , VGG-16 and ResNet50

Custom Classifier

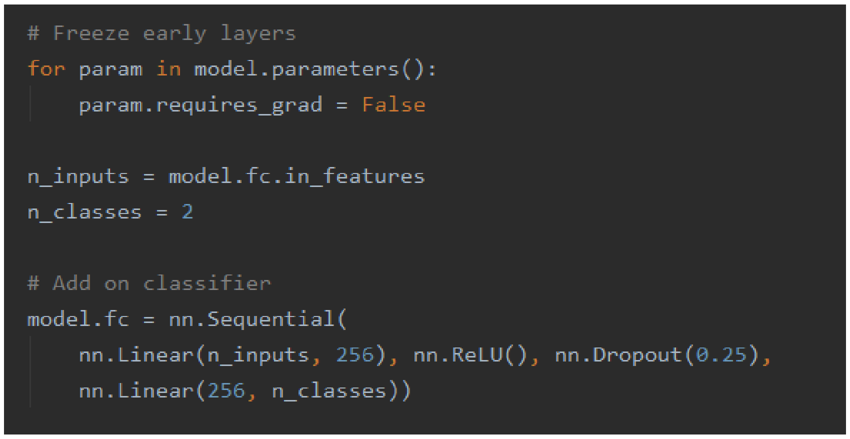
As previously stated, the chosen pretrained models were trained on ImageNet dataset, which has 1000 outputs, Therefore, adding fully connected layers to custom the model to have it fit for our problem. *Figure 7* and *Figure 8* illustrates the code used to freeze early convolutional layers and adding custom fully connected layers.



```
for param in model.parameters():
    param.requires_grad = False
n_inputs = model.classifier[6].in_features
n_classes = 2

# Add on classifier
model.classifier[6] = nn.Sequential(
    nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
    nn.Linear(256, n_classes))
```

Figure 7. Custom Classifiers for VGG-16 and AlexNet



```
# Freeze early layers
for param in model.parameters():
    param.requires_grad = False

n_inputs = model.fc.in_features
n_classes = 2

# Add on classifier
model.fc = nn.Sequential(
    nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.25),
    nn.Linear(256, n_classes))
```

Figure 8. Custom Classifiers for ResNet50

Loss Function

Since the model is to be trained to classify two classes (“Damaged” or “No Damage”). The loss function used is the Cross Entropy loss. Cross entropy increases as the predicted probability diverges from the actual label. The Cross Entropy is calculated over all mini-batches and then averaged. **CrossEntropyLoss** Criterion in pytorch combines **LogSoftmax** and **NLLLoss**.

Optimizer

The optimizer chosen for training is SGD. SGD provides more stable calculations when compared to Adam, however, this required more training time. In addition, momentum rate is set to 0.9. Momentum helps in accelerating gradients in the right direction, thus leads to faster convergence.

Hyperparameter Tuning

Next step, multiple Hyperparameters experiments were conducted to find optimum Batch Size, mini-batch size and Learning rate. *Table 2* shows the final hyperparameters used in training the model. For all models, the modelling started by loading 50 images to the Dataloader. The number of images increased to 300 , However, this caused more time in training where it reached up to 2 hours for 80 epochs while training VGG-16. In addition, for all pretrained model, increasing the number of mini-batches did not achieve an adequate performance.

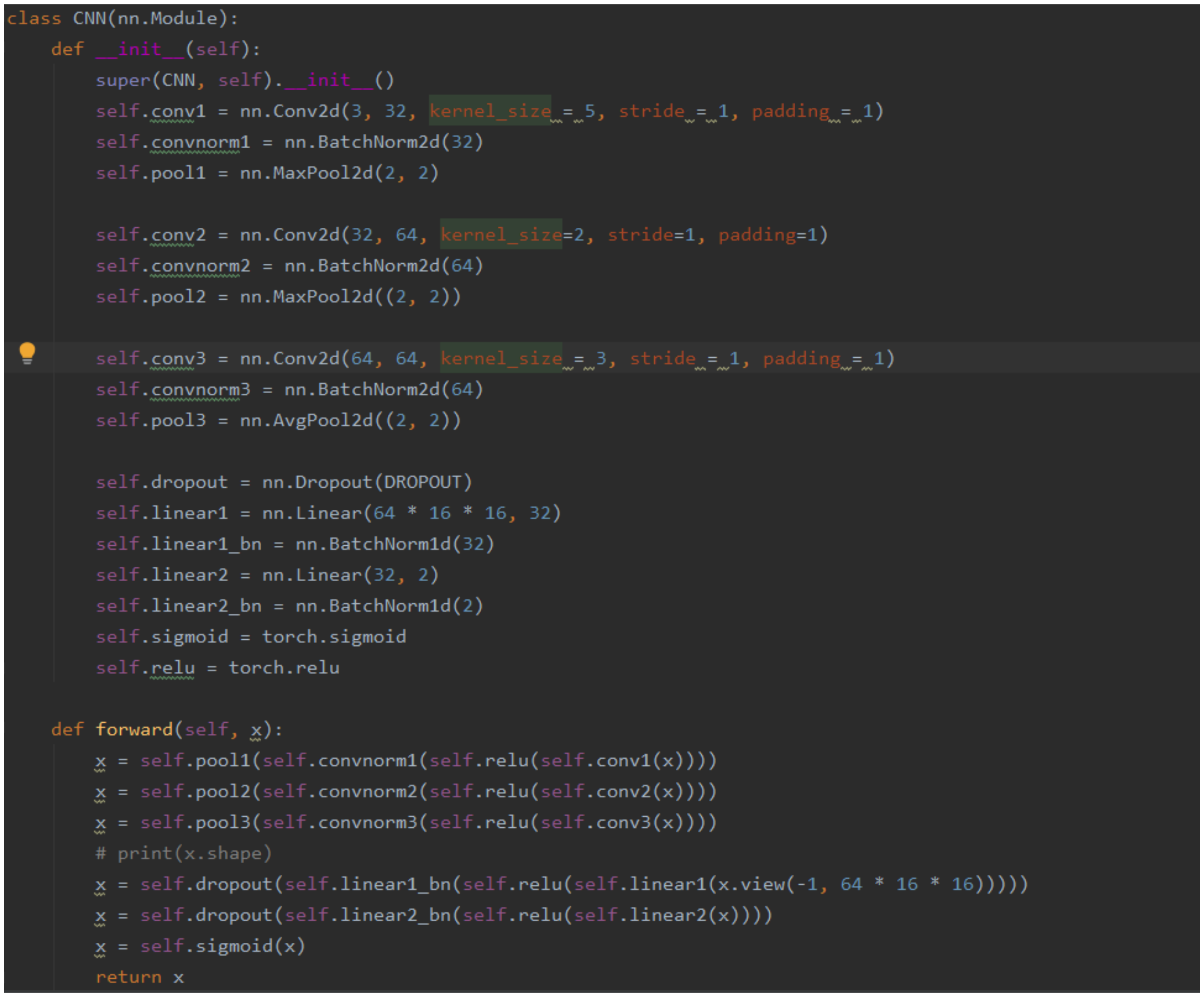
Hyperparameter	VGG-16	ResNet50	AlexNet
Batch size	300	300	300
Mini-batch size	5	10	3
Learning rate	0.00001	0.00002	0.00002

Table 2. Hyperparameter Tuning results for pretrained models

Custom CNN model Experimental setup

Custom model architecture

The custom CNN consists of three building block followed by two fully connected layers. Each block consists of a convolutional layer with different kernel size (5,2 and 3), a Relu activation layer, one batch normalization layer and a pooling layer. *Figure 9* Illustrates the code written to define this network.



```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size = 5, stride = 1, padding = 1)
        self.convnorm1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=1)
        self.convnorm2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d((2, 2))

        self.conv3 = nn.Conv2d(64, 64, kernel_size = 3, stride = 1, padding = 1)
        self.convnorm3 = nn.BatchNorm2d(64)
        self.pool3 = nn.AvgPool2d((2, 2))

        self.dropout = nn.Dropout(DROPOUT)
        self.linear1 = nn.Linear(64 * 16 * 16, 32)
        self.linear1_bn = nn.BatchNorm1d(32)
        self.linear2 = nn.Linear(32, 2)
        self.linear2_bn = nn.BatchNorm1d(2)
        self.sigmoid = torch.sigmoid
        self.relu = torch.relu

    def forward(self, x):
        x = self.pool1(self.convnorm1(self.relu(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.relu(self.conv2(x))))
        x = self.pool3(self.convnorm3(self.relu(self.conv3(x))))
        # print(x.shape)
        x = self.dropout(self.linear1_bn(self.relu(self.linear1(x.view(-1, 64 * 16 * 16)))))
        x = self.dropout(self.linear2_bn(self.relu(self.linear2(x))))
        x = self.sigmoid(x)
        return x
```

Figure 9. Custom CNN Network

Same as the pretrained networks, Image augmentation was performed with images sizes of (128,128,3), Loss function was chosen to be Cross Entropy Loss. In addition, SGD optimizer with momentum value of 0.9. Dropout layers were added after each fully connected layer to prevent overfitting.

Hyperparameter Tuning

Next step, multiple Hyperparameters experiments were conducted to find optimum Batch Size, mini-batch size and Learning rate. *Table 3* shows the final hyperparameters used in training the model. For all models, the modelling started by loading 50 images to the Dataloader. The number of images increased to 200. In addition, increasing the number of mini-batches beyond 10 did not achieve an adequate performance. Various learning rates were experimented and a value of 0.01 achieved best results.

Hyperparameter	Gebril-model
Batch size	200
Mini-batch size	10
Learning rate	0.01

Table 3. Hyperparameters Tuning results for custom model

Results

As previously stated, all models were tested on unbalanced and balanced test sets. The unbalanced test set consists of 8,000 Damaged labels and 1,000 undamaged labels. The balanced test set contains 1,000 images of each label. Accuracy, confusion matrix and F-1 score where used as performance metrics.

Unbalanced Test Set

Figure 10 Illustrates the confusion matrix of all models on the unbalanced test set. *Table 4* shows the metrics calculated based on classification report.

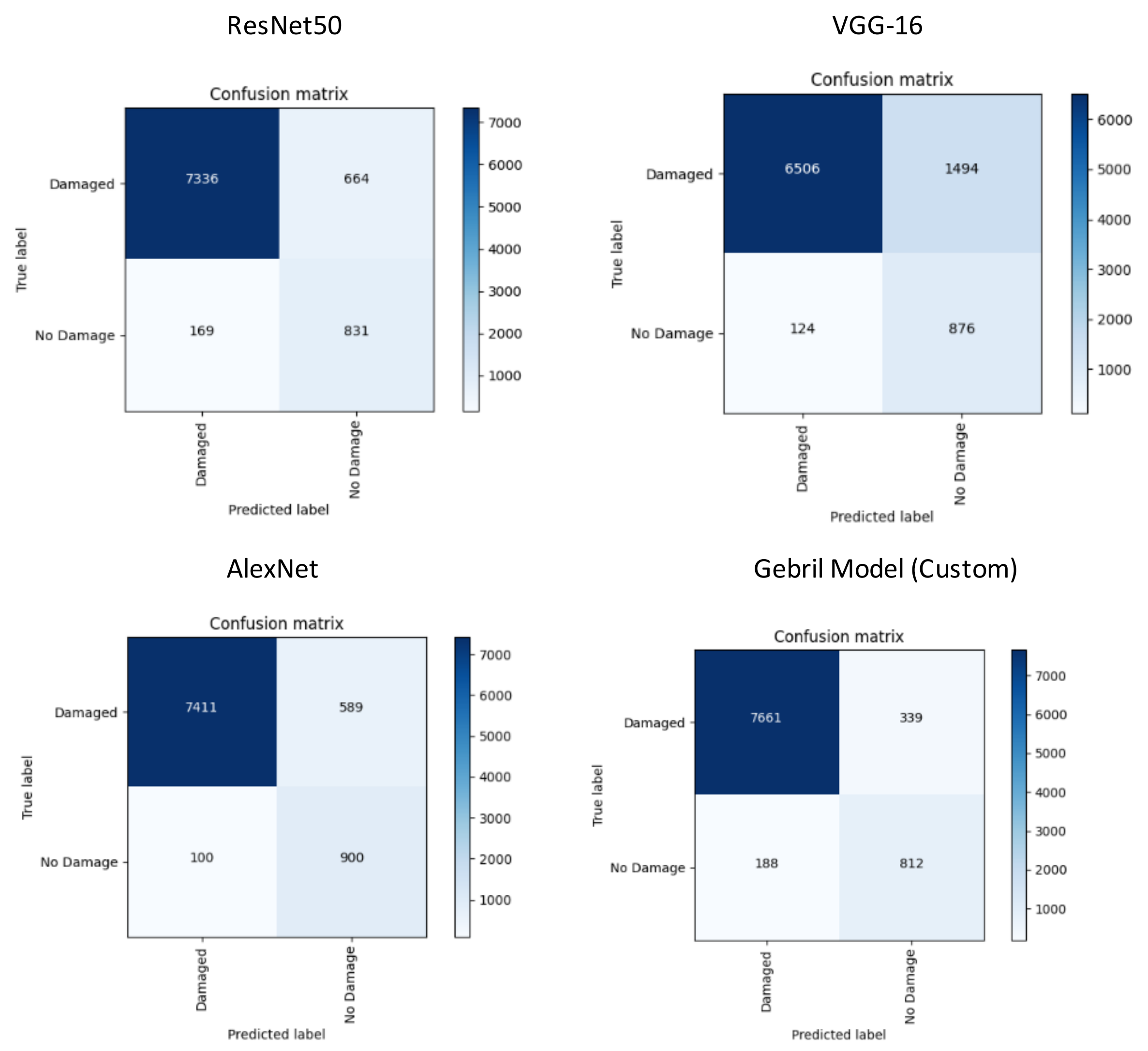


Figure 10. Confusion matrix after testing on unbalanced test set

Model	Accuracy	Precision	Recall	F1-score
Gebril model	94.14%	0.957	0.976	0.8608
AlexNet	92.3%	0.926	0.986	0.839
ResNet50	90.74%	0.917	0.977	0.80
VGG-16	82.02%	0.813	0.981	0.70

Table 4. Models Performance on unbalanced test set

The custom model (Gebril) scored the highest accuracy rate compared to other models with good precision, recall and f-1 score, indicating that the model is performing well in terms of correctly classifying labels. VGG-16 had the lowest performance metrics with a substantial amount when compared to other models. It is important to note that based on the time consumed in training VGG-16 and time constraints, better results can be achieved with more hyperparameters tuning.

Balanced Test Set

Figure 11 Illustrates the confusion matrix of all models on the unbalanced test set. *Table 5* shows the metrics calculated based on classification report.

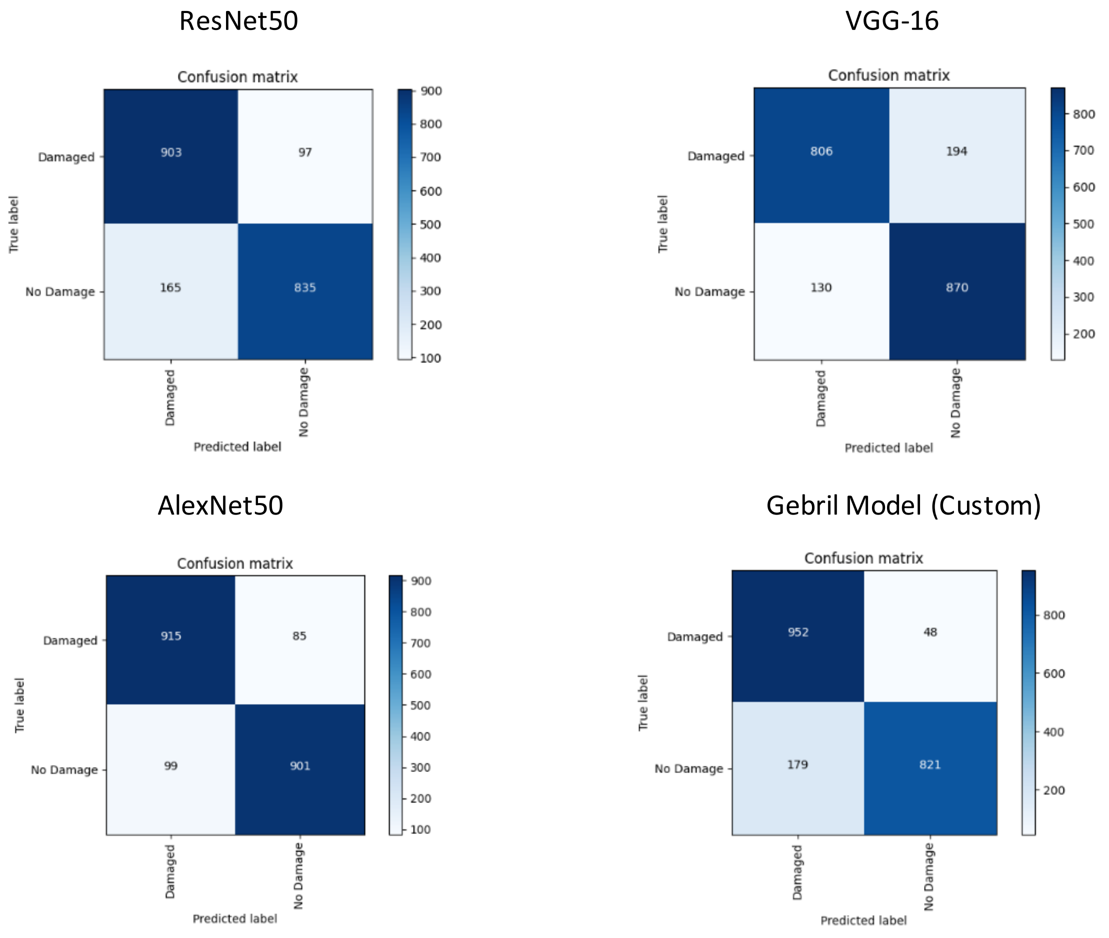


Figure 11. Confusion matrix after testing on balanced test set

Model	Accuracy	Precision	Recall	F1-score
Gebril model	88.65%	0.952	0.84	0.886
AlexNet	90.8%	0.915	0.902	0.97
ResNet50	86.9%	0.903	0.84	0.868
VGG-16	83.8%	0.806	0.861	0.837

Table 5. Models Performance on balanced test set

AlexNet scored the highest accuracy rate compared to other models with good precision, recall and f-1 score, indicating that the model is performing well in terms of correctly classifying labels. VGG-16 had the lowest performance metrics with a substantial amount when compared to other models. An observation here is to that Gebril (Custom) model and ResNet50 did not perform as good as measured on unbalanced set as seen in the obtained recall value. This suggests overfitting. Dropout was initially used to reduce overfitting; different techniques were also used such as L2 Regularization but did not produce an improved performance.

This also can be observed from *Figure 12* and *Figure 13* that illustrates the accuracy and loss over epochs. It is noticed from the accuracy plots that the validation accuracy for models Gebril and ResNet50 suggests underfitting, where its values were higher than training accuracy. Note here that the training accuracy and loss are calculated as average measures of batches while the validations measures are calculated by the end of epoch. Therefore, monitoring the accuracy overtime did not show that. In addition, Experiments by adding more fully connected layers and decreasing dropout did not provide better performance.

Based on the loss plots in *Figure 13*, The gap between training and validation losses needs more examining. It is previously state that the training loss is calculated during epoch and averaged over batch size while validation loss is calculated after epoch. However, further examination is needed. All models have an overall decreased training and validation loss. However, Note AlexNet training curve is much smoother compared to other models where training loss over epochs have more small fluctuations. This might explain the consistent performance of AlexNet model on both tests sets compared to all models specially Gebril (Custom) and ResNet50 models.

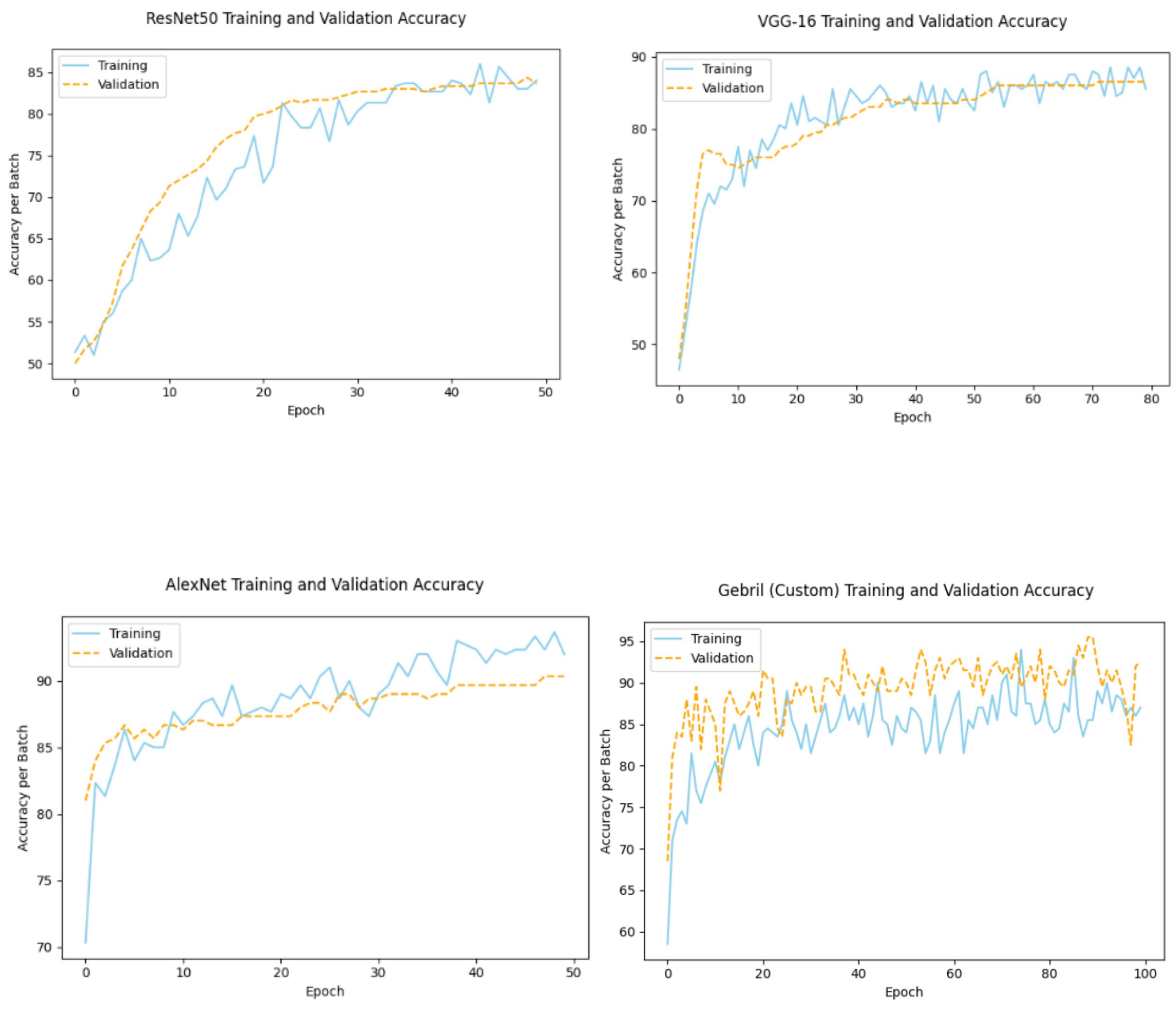


Figure 12. Training and validation accuracy plots

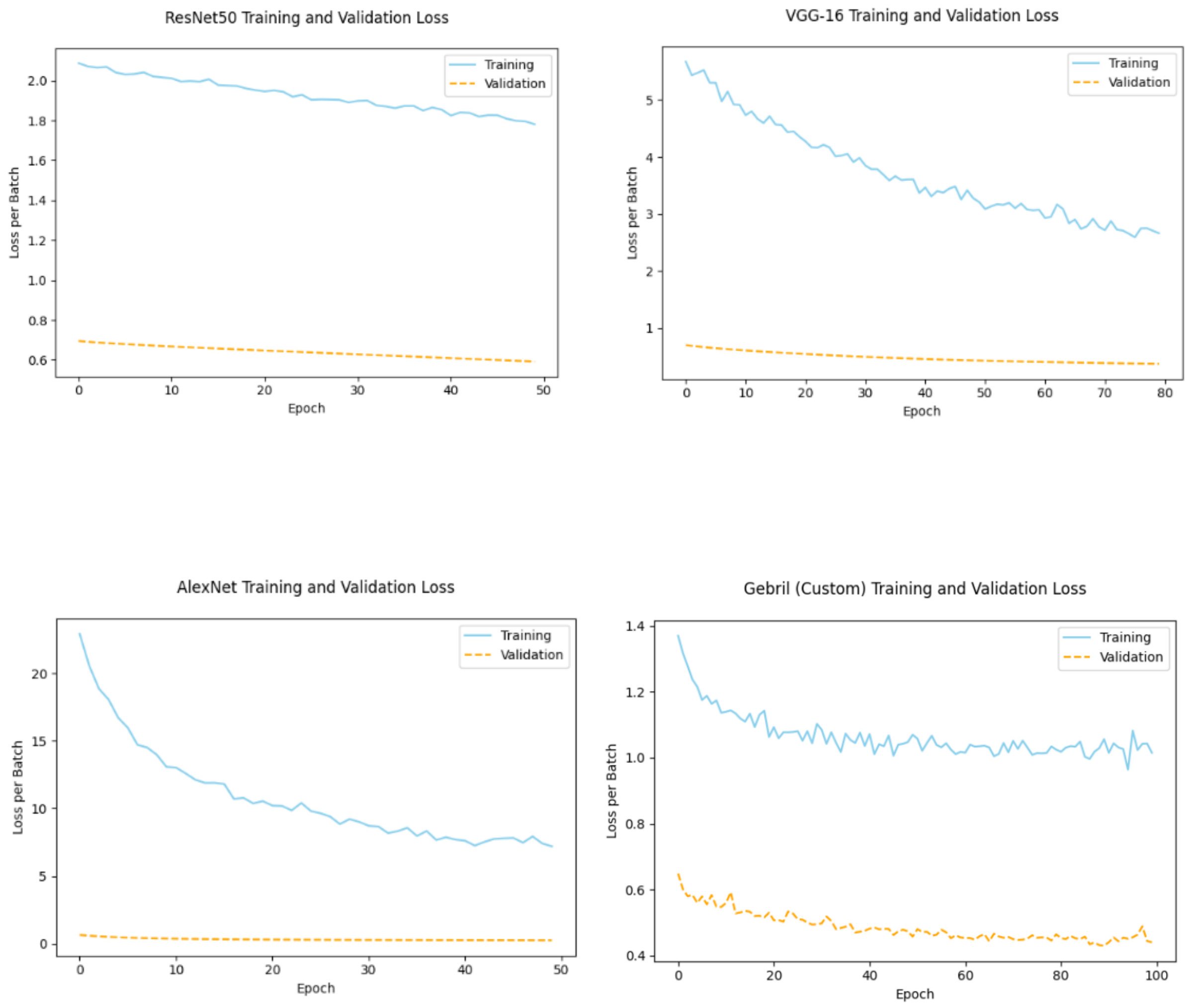


Figure 13. Training and validation loss plots

Conclusion

The relief efficiency to areas affected by natural disasters is greatly affected by the efficiency of damage assessment. This report discussed the automation of detecting flood damages using satellite images of Greater Huston area post Hurricane Harvey in 2017. Initial work was featured in a research paper by Quoc Dung Cao and Youngjun Choe published in the Institute of Electrical and Electronics Engineering Journal (December, 2018). Cao and Choe model's achieved an accuracy higher than 97%. Four different models were trained including three using pretrained model (AlexNet, VGG-16 & ResNet50). All models were tested on unbalanced and balanced test sets. AlexNet model achieved the highest overall accuracy compared to other models.

Future work will include addressing the overfitting and inconsistency of some model performances. Different augmentation and regularization techniques will be performed to achieve better results. In addition, an improved Hyperparameter tuning using various libraries will be conducted. Finally, broadening the scope of the model to include data from other structures such as roads and bridges to make our model more comprehensive.

References

1. Quoc Dung Cao, Youngjun Choe, December 13, 2018, "Detecting Damaged Buildings on Post-Hurricane Satellite Imagery Based on Customized Convolutional Neural Networks", IEEE Dataport, doi: https://dx.doi.org/10.21227/sdad-1e56.

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

3. https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

4. https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e4b1

5. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

6. https://github.com/pytorch/tutorials/blob/master/beginner_source/transfer_learning_tutorial.py

Code Appendix

- https://github.com/AbdulazizGebril/Damaged_Building_Detection_Post_Hurricane