Deep Learning Group Challenge

Hurricane Damage Detector

Jose Araujo, Mehran Chowdhury, Jonas Voßemer, Vincent Zitzewitz

23.06.2023

What is the challenge about?

- Use satellite imagery data to detect damaged buildings after a hurricane

Damaged or not?

Damaged or not?

What is the challenge about?

- Use a data set from a **paper by Cao and Choe (2020)**
- Data provides to us includes **1358 images** (128x128) of training data, on which we applied a **80-20 training-validation** data split
- **12228 test images** that can be used to assess the performance
- The distribution of damages and no damages is **roughly balanced** in both

- Goal defined for us: > **90% accuracy on test data**

Why is this important? (Cao and Cheo, 2020)

- Damage assessment is critical
  - to emergency managers for efficient response
  - resource allocation
- How to gauge the damage extent, quantify the number of damaged buildings?
  - Traditionally done by ground survey
  - But this process can be labor-intensive and time-consuming

- **Can Deep Learning achieve high accuracy and replace the traditional process?**

Approaches

- CNN model(s) from scratch (Jonas and Jose)
  - We used the original paper by Chao and Choe (2020) as a starting point

- Transfer Learning
  - Use Resnet50 as a base model and apply transfer learning (Mehran)
  - Use VGG16 as a base model and apply transfer learning (Vincent)

CNN models from scratch

This is the “best” architecture used by Cao and Choe (2020) → CC2020

Table 1 Convolutional neural network architecture that achieves the best result.

Layer type	Output shape	Number of trainable parameters
Input	3@(150x150)	0
2-D Convolutional 32@(3x3)	32@(148x148)	896
2-D Max pooling (2x2)	32@(74x74)	0
2-D Convolutional 64@(3x3)	64@(72x72)	18,496
2-D Max pooling (2x2)	64@(36x36)	0
2-D Convolutional 128@(3x3)	128@(34x34)	73,856
2-D Max pooling (2x2)	128@(17x17)	0
2-D Convolutional 128@(3x3)	128@(15x15)	147,584
2-D Max pooling (2x2)	128@(7x7)	0
Flattening	1x6272	0
Dropout	1x6272	0
Fully connected layer	1x512	3,211,776
Fully connected layer	1x1	513

Note: The total number of trainable parameters is 3,453,121. C@($A \times B$) is interpreted as that there are a total of C matrices of shape ($A \times B$) stacked on top of one another to form a three-dimensional tensor. 2-D Max pooling layer with (2×2) pooling size means that the input tensor’s size will be reduced by a factor of 4.

In addition they applied

- Data augmentation
  - Rotation, horizontal flipping, …
- Dropout before the fully connected layer was 50%
- L2 regularization in the fully connected layer with lambda 10^-6

CNN models from scratch

1. CC2020 **without** data augmentation, dropout, and l2 regularization

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 128, 128, 3)       0

 conv2d (Conv2D)             (None, 128, 128, 32)      896

 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 4,436,161
Trainable params: 4,436,161
Non-trainable params: 0
_________________________________________________________________
```

	precision	recall	f1-score	support
no_damage	0.94	0.85	0.89	6333
damage	0.85	0.94	0.89	5895
accuracy			0.89	12228
macro avg	0.90	0.90	0.89	12228
weighted avg	0.90	0.89	0.89	12228

	0	1
0	5390	943
1	357	5538

CNN models from scratch

2. CC2020 **with** data augmentation, dropout and l2 regularization

- Trying to mimic CC2020 as close as possible in terms of architecture but also data augmentation and parameter choices

```
data_augmentation = tf.keras.Sequential([
            layers.experimental.preprocessing.Rescaling(1./255, input_shape=(128, 128, 3)),
            layers.experimental.preprocessing.RandomRotation(0.2),
            layers.experimental.preprocessing.RandomFlip("horizontal"),
            layers.experimental.preprocessing.RandomTranslation(0.1, 0.1),
            layers.experimental.preprocessing.RandomZoom(0.2),
        ])
```

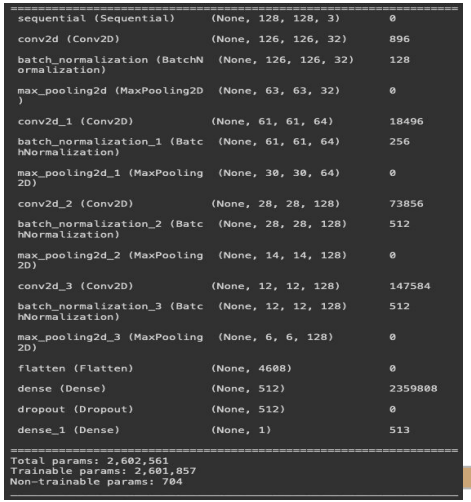
	precision	recall	f1-score	support
no_damage	0.92	0.92	0.92	6333
damage	0.91	0.92	0.91	5895
accuracy			0.92	12228
macro avg	0.92	0.92	0.92	12228
weighted avg	0.92	0.92	0.92	12228

	0	1
0	5819	514
1	497	5398

CNN models from scratch

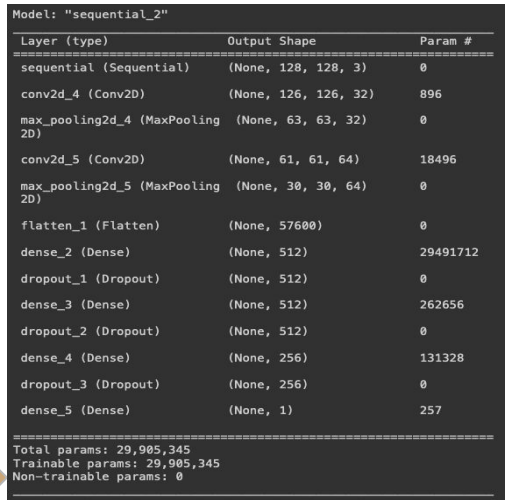
3. CC2020 **with** data augmentation and dropout

- Reducing the number of convolution layers and increasing dense layers complexity



Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	147584
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 512)	2359808
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

Total params: 2,602,561
Trainable params: 2,601,857
Non-trainable params: 704



Model: "sequential_2"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 128, 128, 3)	0
conv2d_4 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_5 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_5 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten_1 (Flatten)	(None, 57600)	0
dense_2 (Dense)	(None, 512)	29491712
dropout_1 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 1)	257

Total params: 29,905,345
Trainable params: 29,905,345
Non-trainable params: 0

CNN models from scratch

3. CC2020 **with** data augmentation and dropout

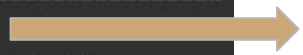
- Reducing the number of convolution layers and increasing dense layers complexity

	precision	recall	f1-score	support
0	0.98	0.83	0.90	6333
1	0.84	0.98	0.90	5895
accuracy			0.90	12228
macro avg	0.91	0.90	0.90	12228
weighted avg	0.91	0.90	0.90	12228

Accuracy: 90.10%

Precision: 91.13%

Recall: 90.10%



	precision	recall	f1-score	support
0	0.80	0.88	0.84	6333
1	0.86	0.76	0.80	5895
accuracy			0.82	12228
macro avg	0.83	0.82	0.82	12228
weighted avg	0.83	0.82	0.82	12228

Accuracy: 82.26%

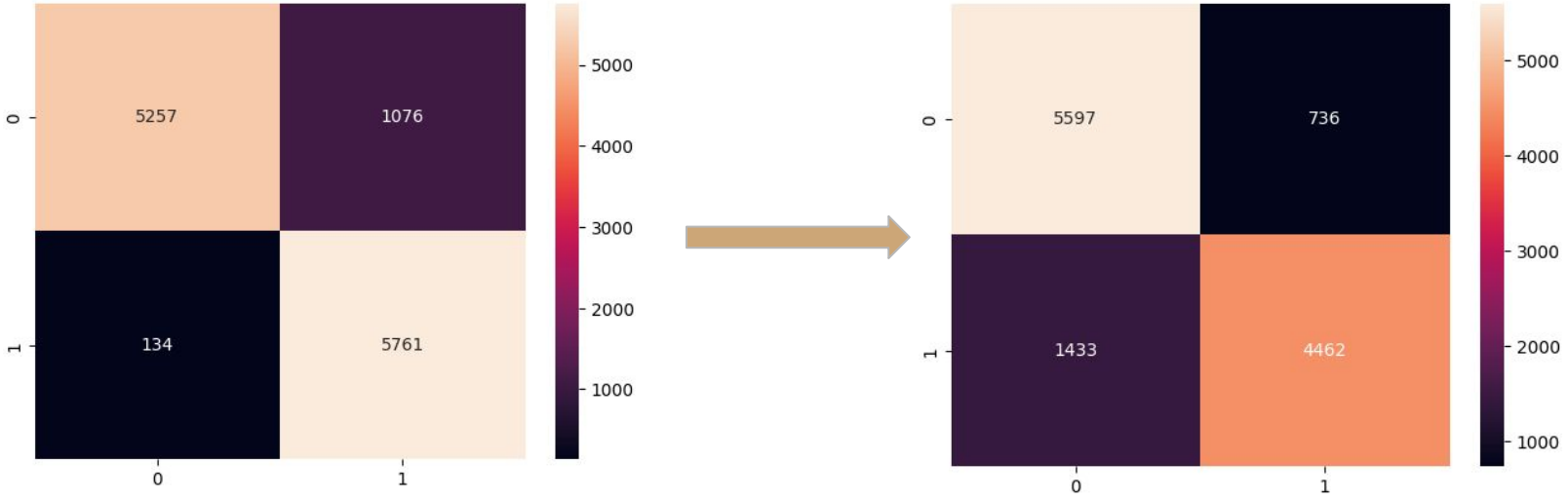
Precision: 82.62%

Recall: 82.26%

CNN models from scratch

3. CC2020 **with** data augmentation and dropout

- Reducing the number of convolution layers and increasing dense layers complexity



Transfer Learning: Resnet50

Resnet 50 model **with** data augmentation

conv5_block3_out (Activation)	(None, 4, 4, 2048)	0	['conv5_block3_add[0][0]']
global_average_pooling2d_16 (GlobalAveragePooling2D)	(None, 2048)	0	['conv5_block3_out[0][0]']
dense_47 (Dense)	(None, 256)	524544	['global_average_pooling2d_16[0][0]']
dense_48 (Dense)	(None, 1)	257	['dense_47[0][0]']

Total params: 24,112,513
Trainable params: 524,801
Non-trainable params: 23,587,712

Confusion Matrix:

```
[[5978  355]
 [ 396 5499]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.94	0.94	6333
1	0.94	0.93	0.94	5895
accuracy			0.94	12228
macro avg	0.94	0.94	0.94	12228
weighted avg	0.94	0.94	0.94	12228

Accuracy: 0.9385835786719006

Transfer Learning: Resnet50

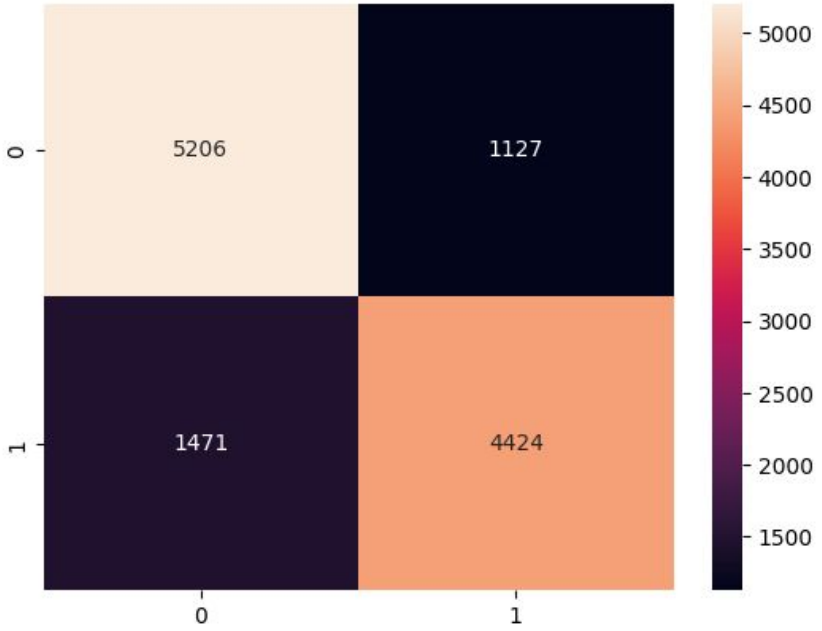
Problems replicating results! Data augmentation?

	precision	recall	f1-score	support
0	0.78	0.82	0.80	6333
1	0.80	0.75	0.77	5895
accuracy			0.79	12228
macro avg	0.79	0.79	0.79	12228
weighted avg	0.79	0.79	0.79	12228

Accuracy: 78.75%

Precision: 78.80%

Recall: 78.75%



Transfer Learning: VGG16

Base Model

Preprocess
Conv2D
Pool
Flatten
Dense x2

x3
x2

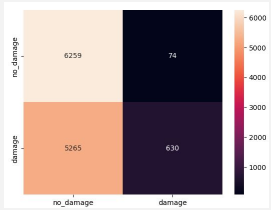
Optimizer: Adam

F1 Score: 0.88

Variation: SGD

Preprocess
Conv2D
Pool
Flatten
Dense x2

x3
x2



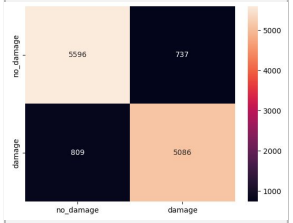
Optimizer: SGD

F1 Score: 0.56

Variation: More layers

Preprocess
Conv2D
Pool
Flatten
Dense x2

x6
x2



Optimizer: Adam

F1 Score: 0.87

Transfer Learning: VGG16

Base Model

- Preprocess
- Conv2D
- Pool
- Flatten
- Dense x2

x3

x2

Optimizer: Adam

F1 Score: 0.88

VGG16 Model

- Preprocess
- Conv2D
- Conv2D
- Pool
- Conv2D
- Conv2D
- Conv2D
- Pool
- Flatten
- Dense x2

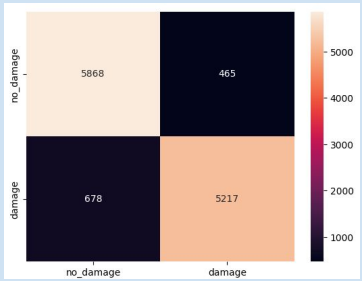
x2

x3

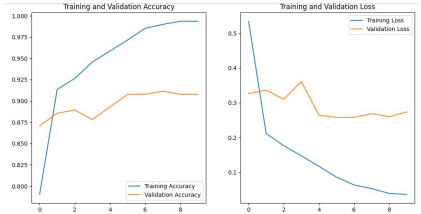
x2

Optimizer: Adam

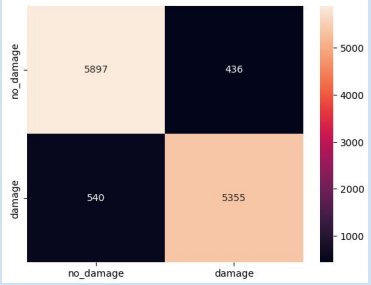
Frozen: All VGG Layers



F1 Score: 0.91



Frozen: No VGG Layers



F1 Score: 0.93

Conclusion

- One needs to be extra careful when working with small dataset and large/complex models

- Data augmentation helped improving the performance, but also one needs to be careful

- Hard to replicate (original/colleagues) results using similar models