Structural Damage Classification with Machine Learning

Minnie Ho[1], Jorge Troncoso[2]

[1]minnie.ho@intel.com, Intel Corporation
[2]jatron@google.com, Google LLC

Stanford
CS 229 Final Project

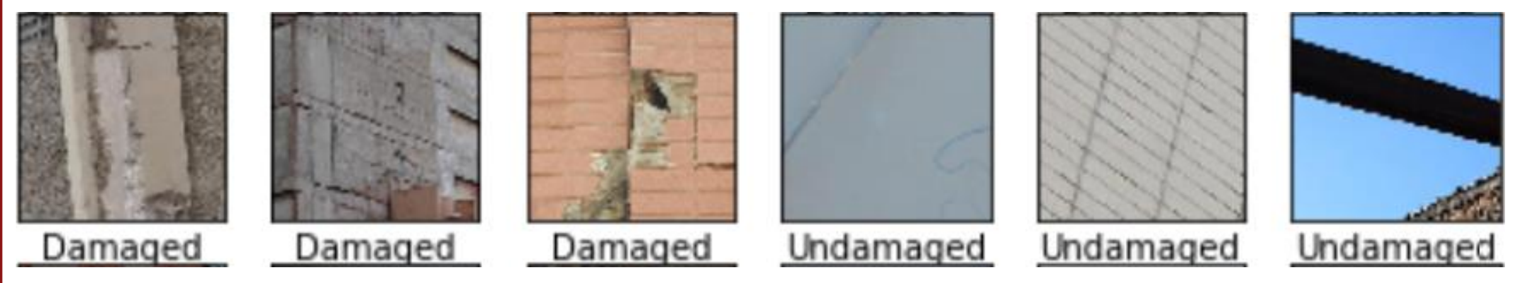
Objective

- Motivation: assess seismic risk of a structure or gather statistics within an area of damage after an earthquake.
- In particular:
  - Build effective image classification models using a variety of machine learning techniques
  - Accurately classify structural damage given an image

Dataset

- 5913 images (224x224 RGB): 2727 damaged (46%), 3186 undamaged (54%), 1479 unlabeled [Ref 1]
- Image characteristics:
  - Mixture of close-ups (section of a wall) and wide-shots (an apartment building)
  - Non-relevant objects included (e.g., people, curtains, telephone lines, tree branches)
  - Mixture of damage type ranging from cracks to leveled buildings
  - Some images difficult to label (previously repaired damage, paint or mortar cover-up, blur); several images incorrectly labeled

Figure 1: Example Images



Preprocessing of Data

- Scaling (normalization) and centering (zero mean) of the images
- No effort to reduce blur, de-noise, or fix incorrect labels (possible future steps)
- Ensured that ratio of damaged vs. undamaged stayed constant (46/54) regardless of training sample size.

Compute Resources

- We used a Google Cloud Deep Learning VM instance for most simulation runs, with optimized Tensorflow (using Intel MKL and NVIDIA CUDA), a NVDIA P100 GPU and Intel Skylake 8-core CPU.
- We discovered an instance optimized for NVDIA was faster on CNNs, but instance optimized for Intel was faster for scipy.

References

[1] Pacific Earthquake Engineering Center. 2018. *PEER Hub ImageNet Challenge*. https://apps.peer.berkeley.edu/phichallenge/detection-tasks/

[2] Scikit-learn. 2007. https://scikit-learn.org/

[3] TensorFlow for Poets. https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0

Acknowledgments

Guidance from Fantine Huot and Mark Daoust gratefully acknowledged.

Model Processing and Results

- Began with three classical machine learning models: (Logistic Regression, K-means, and SVM) using sci-kit learn [Ref 2]
- Results of training and validation accuracies shown in Table 1
- Examples of Convolutional Neural Networks (CNN) such as MobileNetv1.0 and Inceptionv3 performed best, as expected
- Moved to Convolution Neural Networks using Tensorflow-for-Poets [Ref 3]
- Performed diagnostics to guide next steps
- Focused on Inceptionv3

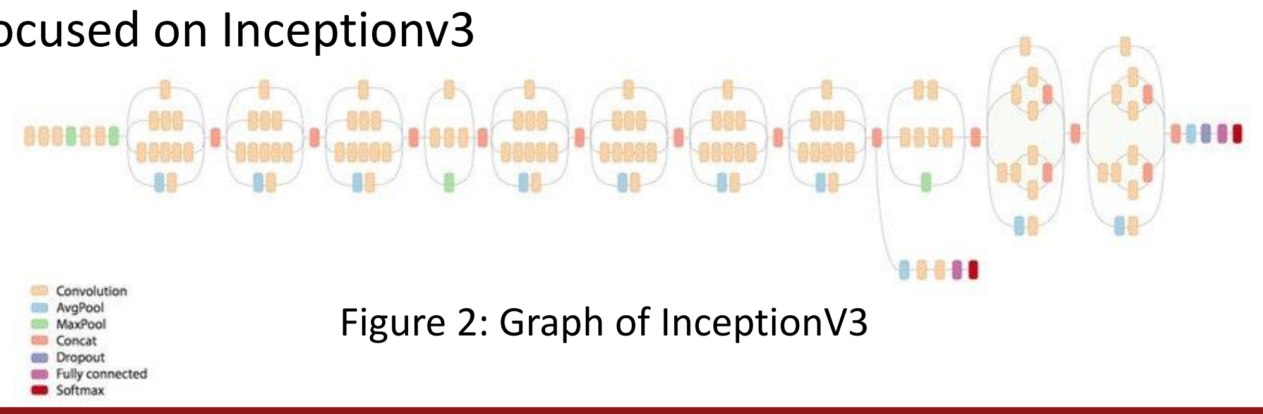


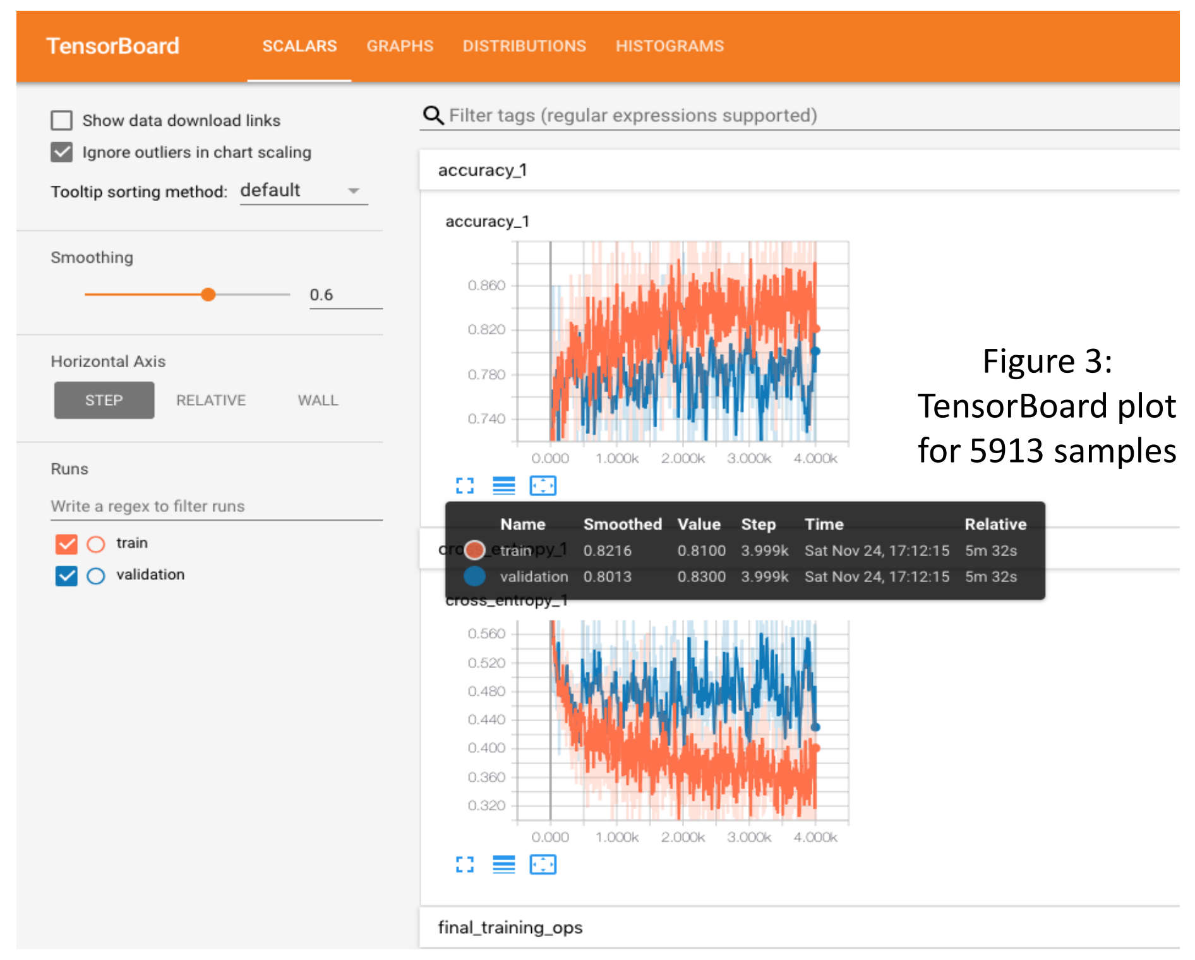
Figure 2: Graph of InceptionV3

Model	Training Accuracy	Validation Accuracy
K-nearest neighbors vote	67.4%	53.7%
Logistic Regression	99.4%	54.7%
SVM	99.8%	50.7%
MobileNetv1.0 (CNN) (tensorflow-for-poets)	80.0%	67.7%
InceptionV3 (CNN) (tensorflow-for-poets)	81.0%	83.0%
SVM trained on activations from layer 288 in InceptionV3	95.0%	75.0%
InceptionV3 (Keras)	100.0%	74.2%
InceptionV3 (Keras + data augmentation)	77.2.0%	72.8%

Table 1: Models and Accuracies

Diagnostics (part 1)

- Computed bias vs variance curve (see plot below)
- Set-up: Tensorflow-for-Poets. Transfer learning using Inceptionv3 CNN network trained on ImageNet, adding a fully connected layer and softmax. GradientDescent with a learning rate of 0.01 (Gradient Descent led to noisy accuracy plots below, hence we later switched to AdamProp). Batch size of 100 with 4000 iterations.
- Clearly a bias issue (although the final accuracies could also be higher). This pointed to training the CNN on additional layers.



Name	Smoothed	Value	Step	Time	Relative
train	0.8216	0.8100	3.999k	Sat Nov 24, 17:12:15	5m 32s
validation	0.8013	0.8300	3.999k	Sat Nov 24, 17:12:15	5m 32s

Figure 3: TensorBoard plot for 5913 samples

Bias vs Variance curve

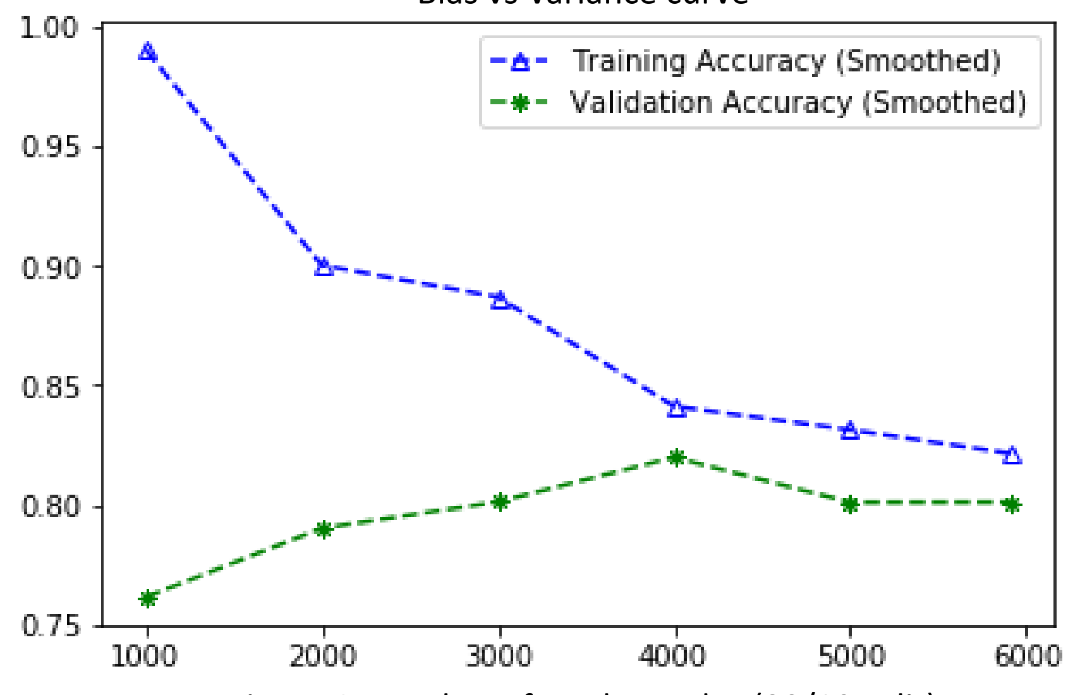


Figure 4. Number of total samples (90/10 split)

Diagnostics (part 2)

- Used trained Inceptionv3 CNN network to predict 1000 test images, using 4000 images to train.
- Confusion matrix: $\begin{bmatrix} 448 & 86 \\ 117 & 349 \end{bmatrix}$
- Checked manually through several hundred images to determine patterns in correct images, false negatives, and false positives.

Figure 5: False Positives (Index, Confidence Probability)

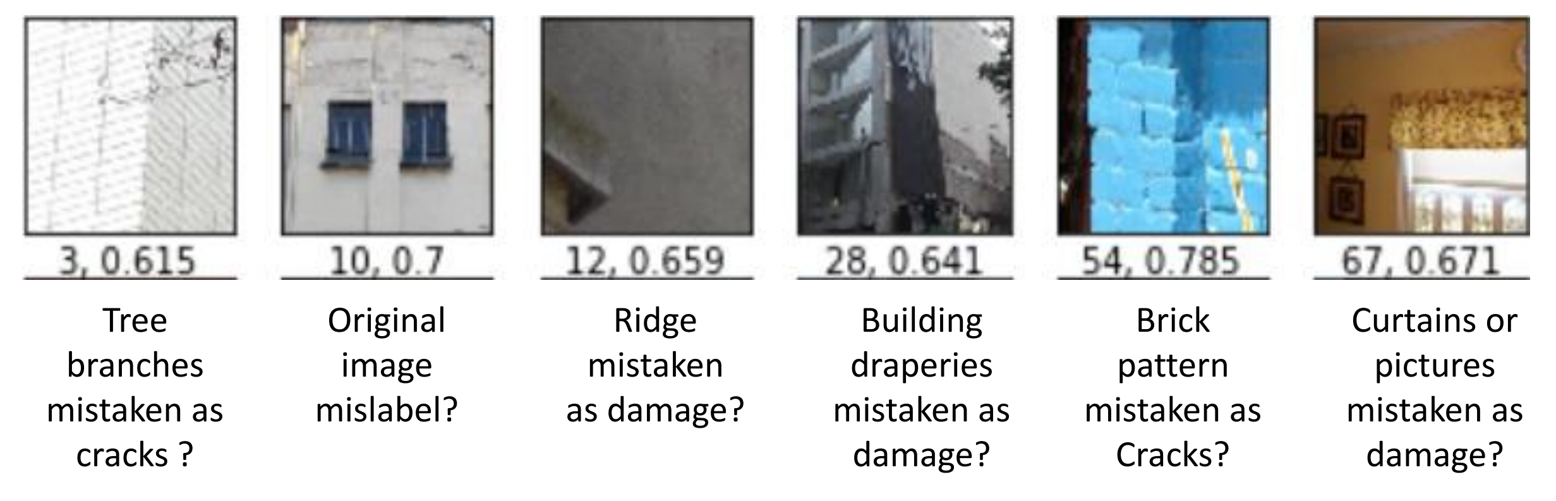
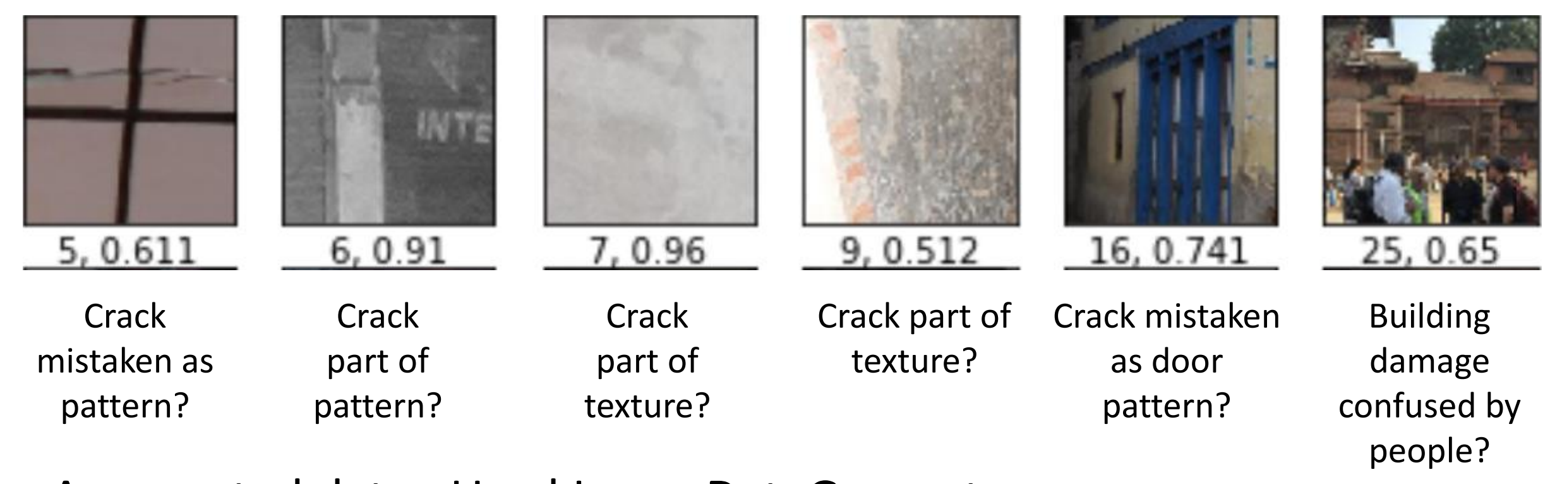
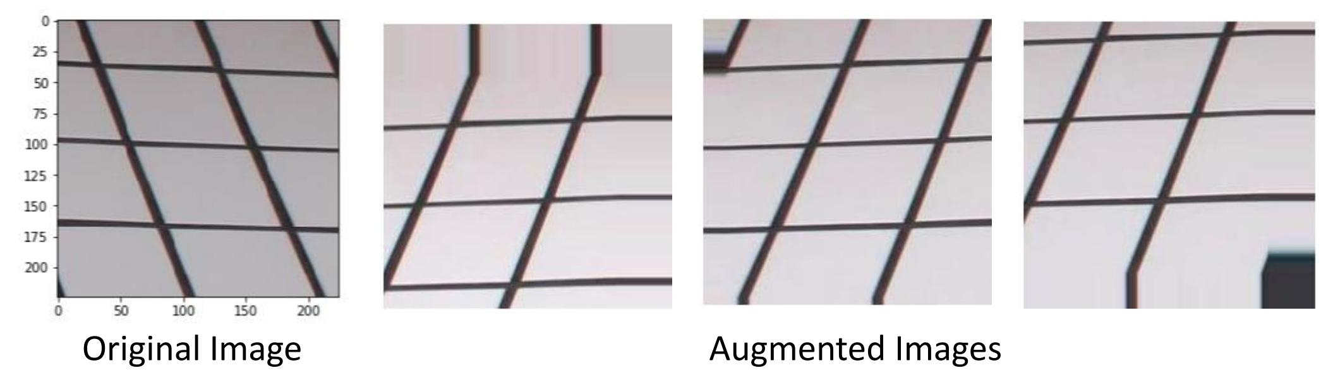

3, 0.615
Tree branches mistaken as cracks ?
10, 0.7
Original image mislabel?
12, 0.659
Ridge mistaken as damage?
28, 0.641
Building draperies mistaken as damage?
54, 0.785
Brick pattern mistaken as Cracks?
67, 0.671
Curtains or pictures mistaken as damage?

Figure 6: False Negatives (Index, Confidence Probability)



- Augmented data: Used ImageDataGenerator
  Width_shift=0.2, height_shift=0.2, horizontal_flip
  Cons: Augmented images are still highly correlated with original images
  Augmented images may not be valid structures

Figure 7: Original and Augmented Images



Putting it all together

- We focused on Inceptionv3 (CNN). Inceptionv3 has 310 layers. Layer 300 is a convolutional layer with 393,216 parameters. Layers 301-310 have 512 trainable parameters.
- The bias vs. variance curve has a bias issue when a fully-connected layer is added to Inceptionv3 (Figure [4]).
- Experiments with training more layers of Inceptionv3 to help with bias. Moved to Keras rather than Tensorflow for Poets.
- Found overfitting issues with move to Keras (Figure [8]).
- We find that we can reduce the variance by feeding slightly different versions of the training images on each iteration. These images were obtained by randomly flipping and shiting images in our training set, so the model never saw the exact same image more than once, making it harder to overfit (Figure [9])

Figure 8

InceptionV3 (Keras)

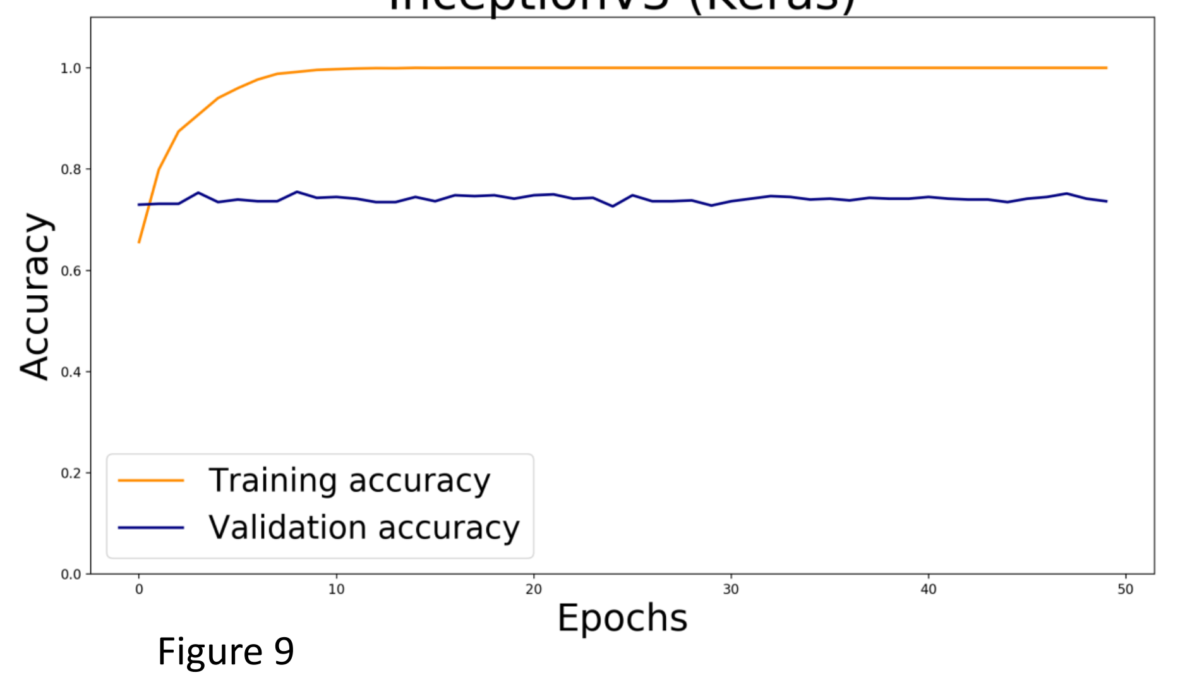
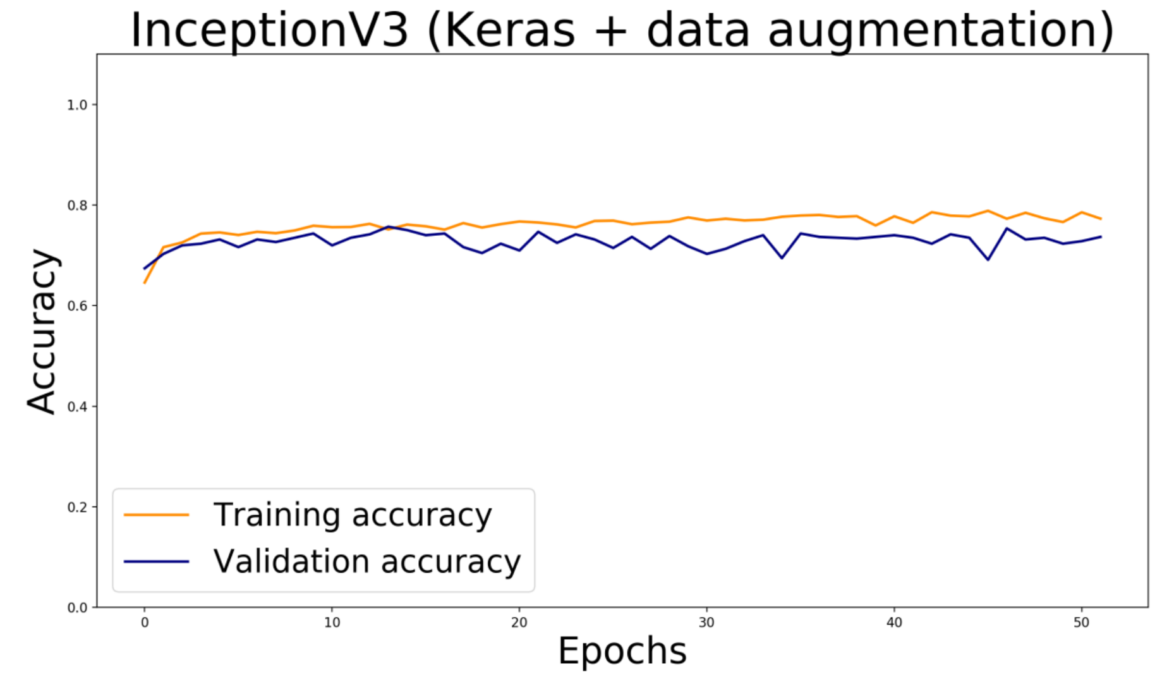


Figure 9

InceptionV3 (Keras + data augmentation)



Conclusions

- Convolution Neural Networks performed the best in our model.
- Bias could be managed by training more layers of the CNN
- Training more layers of the CNN can lead to overfitting
- Overfitting can be managed by adding random images to the data

Next Steps and Future Work

- More controlled experimentation to manage bias vs variance
- Improve validation accuracy by managing the data
  - Check mislabeled data
  - Add actual images similar to false positives or false negatives
  - Cropping irrelevant features
  - Understand differences in texture/pattern vs. damage
  - Wide-angle vs. close-up and effect on classification
- Ensemble averaging of different models