Structural Damage Image Classification

Minnie Ho
Intel Corporation
minnie.ho@intel.com

Jorge Troncoso
Google LLC
jatron@google.com

Abstract

Using a training set provided by the Pacific Earthquake Engineering Research (PEER) Center, we build a classifier to label images of structures as damaged or undamaged using a variety of machine learning techniques: K-nearest neighbors, logistic regression, SVM, and convolutional neural networks (CNN). We find that when compared to classical machine learning techniques, the performance of a CNN is best on our data set. We evaluate the mistakes made by our classifiers, and we tune our models using information gleaned from learning curves. We find that our best performing model, which uses transfer learning using Inceptionv3 trained on ImageNet with an added fully-connected layer and softmax, has a test accuracy of 83%.

1 Introduction

The Pacific Earthquake Engineering Research (PEER) Center has provided image datasets that can be used to classify structures in terms of damage [1]. The goal is to solicit image classification models to establish automated monitoring of the health of a structure using computer vision. In particular, it is desirable to quickly assess the seismic risk of a building in a region prone to earthquakes and to gather statistics on the built environment within a geographic region after an earthquake has occurred.

The input to this project consisted of 5913 images. Of this set of images, 3186 images were labeled as "undamaged" or "0" (54%), and 2727 images were labeled as "damaged" or "1" (46%). Each image includes 224 by 224 eight-bit RGB pixels. We split the images into the following sets:

90% for training: 2870 undamaged and 2451 damaged. (46% are damaged)
10% for validation: 316 undamaged and 276 damaged. (47% are damaged).

We decided not to set aside images for testing because of the limited number of samples, although if we were to eventually submit to an academic journal, we would need to be more rigorous in this regard.

We used several models: three classical machine learning models, several variations of two deep learning models (MobileNet and InceptionV3 convolutional neural networks), and one model which combined classical and deep learning techniques.

The primary output of our classifier models is the accuracy as determined by the number of correctly predicted images over the total number of predicted images.

2 Related Work

There are few references on image classification of damaged buildings. One good survey paper on structural image classification is [2]. Some of the ideas in this survey paper (such as transfer learning) we used on our dataset as well. However, most of our references are not specific to structural images. For the classical machine learning algorithms and the convolutional neural networks, we began with [3] and [4]. The original papers on MobileNet [9] and Inceptionv3 [10] were also illuminating. We also observed that on ImageNet [11], there are 1190 images of structures out of 14,197,122 images (0.008%), so previously trained models trained on ImageNet did not have weights well-optimized for our data set.

3 Dataset and Features

A few examples from our dataset are shown in Figure 1. In our dataset, we had images that ranged from close-ups (a small section of a wall) to wide-shots (an entire apartment building). Non-relevant objects were included (e.g., people, curtains, telephone lines, tree branches). The type of damage ranged from cracks to leveled buildings. Some images were difficult to label visually (previously repaired damage, paint or mortar cover-up, blur), and some images were incorrectly labeled.

Figure 1: Example Structural Images

We normalized the images so each of the 224x224 8-bit RGB pixels (x) was in the range [-1,1). This was done by setting each pixel (x) to: $x = \left(\frac{x}{128}\right) - 1$. For k-nearest neighbors, logistic regression, and support vector machine we also scaled and flattened the pictures before feeding them into the models.

4 Methods

We built six models: three classical machine learning models (K-nearest neighbors with k=5, logistic regression, support vector machine with RBF kernel), two deep learning models (MobileNetv1.0 and InceptionV3 convolutional neural networks), and one model which combined classical and deep learning techniques (support vector machine based on activations earlier in the InceptionV3 network). The performance of each of these models is summarized in the Results section.

4.1 K-nearest neighbors

In K-nearest neighbors, "an unlabeled vector is classified by assigning the label which is most frequent among the k training samples nearest to that query point" [5]. Due to the suboptimal results achieved with k=5, we did not spend additional time tuning the k parameter.

4.2 Logistic Regression

In logistic regression, we use the sigmoid function to estimate the probability that an image belongs to a certain class. This sigmoid function is parametrized by a vector θ, which is obtained by maximizing the log-likelihood. Our logistic regression model included L2 regularization with $C = 1.0$. Due to the suboptimal results achieved by this model, we did not spend additional time tuning the regularization parameters.

4.3 Support Vector Machine

During training, support vector machines try to find the "maximum-margin hyperplane" that divides data points with different labels. "Supports vector machines can also efficiently perform non-linear classification using what is called the kernel trick, implicitly mapping the inputs into high-dimensional feature spaces" [6].

Our support vector machine model performed non-linear classification using the radial basis function kernel, which is defined by the formula below.

$$K(x, x') = e^{-\gamma\|x-x'\|^2}$$

We set the penalty parameter C of the error term to 1.0 and the kernel coefficient γ for the RBF kernel to 0.001. Due to the suboptimal results achieved by this first model, we did not do further tuning of these parameters.

4.4 MobileNetV1 and InceptionV3

MobileNetV1 and InceptionV3 are two convolutional neural network (CNN) architectures designed for image recognition tasks. MobileNetV1 is a lighter, lower-latency neural network designed for use on mobile devices, while InceptionV3 is a heavier architecture, which tends to achieve better performance. A diagram of the InceptionV3 architecture is shown below. To give an idea of the complexity of the

network, we note that the top layer [311] of the Inceptionv3 network includes 1000 parameters. Layers [310] includes 102,402 parameters, Layers [301-310] have 512 parameters, and Layer 300 is a convolutional layer with 393,216 parameters. These observations are relevant when managing bias and variance (overfitting).

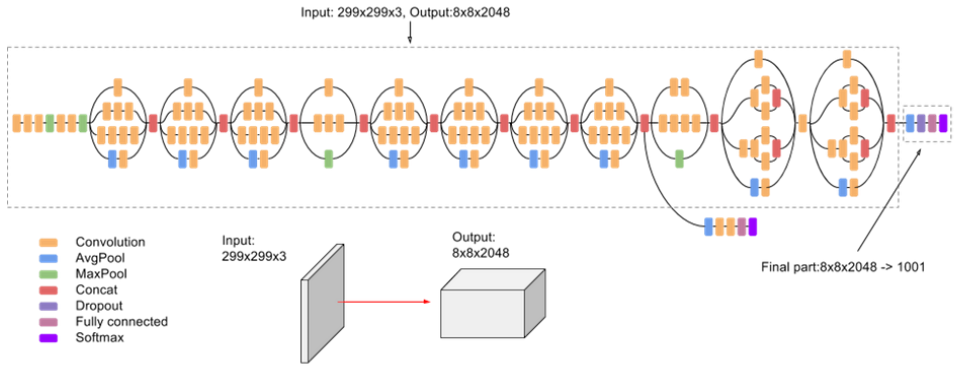

Figure 2: Inceptionv3 Network

Since we only had a few thousand images, training these networks from scratch would surely cause overfitting, so instead, we downloaded pre-trained versions of these models using Tensorflow (with weights optimized to classify images in the ImageNet dataset), froze the weights of most of the layers of the pre-trained networks, and trained a new fully connected layer with a sigmoid or softmax activation placed on top of each of the pre-trained networks. This is a common technique used in machine learning known as transfer learning [8].

4.5 Support Vector Machine Based on Activations Earlier in the InceptionV3 Network

Since our dataset was quite different from the ImageNet dataset, the features extracted at the top of the InceptionV3 network were probably not optimized for our application, so we thought we might be able to achieve better performance by building an SVM classifier based on activations earlier in the InceptionV3 network, which contains more general features.

This was achieved by feeding the pretrained InceptionV3 network all of our images, computing the output of the 288th later (for reference, the InceptionV3 network has 311 layers), and using these outputs as features for an SVM classifier. Here, we also implemented model selection to find the optimal kernel coefficient gamma of the RBF kernel, as shown in Figure 3.

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

All of the code used in this project (including many experiments whose results we did not include in this report, due to lack of space) is available in our GitHub repository:
https://github.com/jatron/structural-damage-recognition.

5 Experiments and Results

5.1 Experimental Results

The performance achieved by each of our models is summarized in the table below.

Model	Train Accuracy	Validation/Test Accuracy
K-nearest neighbors (k=5)	67.4%	53.7%/52.9%
Logistic Regression	99.4%	54.7%
SVM with RBF kernel	99.7%	50.7%
MobileNetV1	80.0%	67.0%
InceptionV3	81.0%	83.0%

SVM based on activations from layer 288 of InceptionV3	95.0%	75.0%

Table 1: Summary of Accuracies for Classification Models

It is not surprising that the models based on CNNs performed the best, since the parameters could best take advantage of the spatial information in the images. We note however, that the mixed network (SVM plus Inceptionv3) also did well; after tuning the kernel coefficient gamma of the RBF kernel, we were able to achieve 75% validation accuracy and 95% training accuracy with this model.

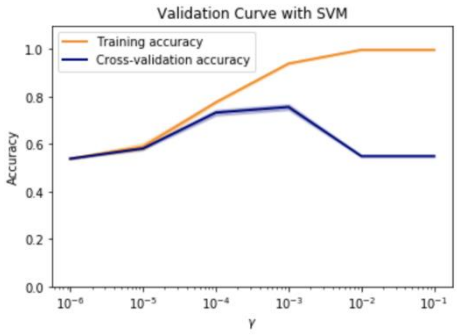

Figure 3: Tuning of the γ parameter for Inceptionv3-SVM

5.2 Bias versus Variance

As mentioned earlier, we had applied transfer learning in Tensorflow to baseline Inceptionv3 model originally trained using ImageNet, adding a fully-connected and softmax layer, similar to [4]. Using a gradient descent algorithm with a learning rate of 0.01, we obtain the results shown in Figure 4 below. Although the final validation accuracy could be higher, we see that bias clearly dominates the performance. As a result, we decided to add more parameters to combat the bias by training additional layers of the Inceptionv3 model.

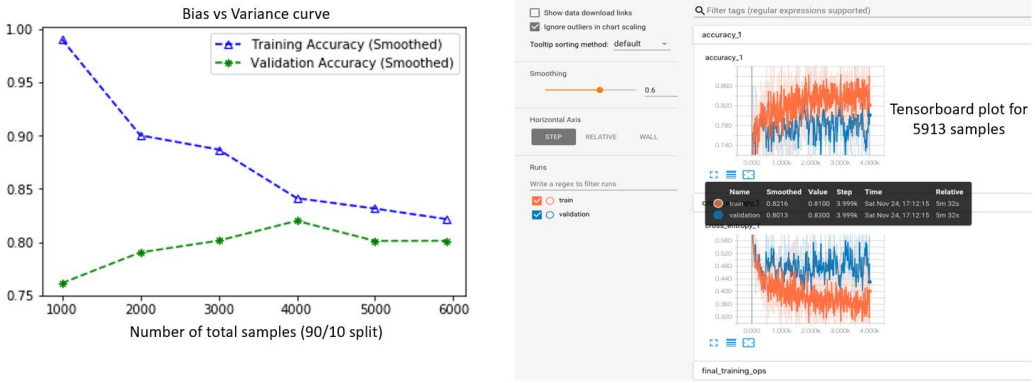

Figure 4: Learning curve for retrained Inceptionv3 (left) and example Tensorboard plot (right)

5.3 Misclassified Images and Data Augmentation

We find that by using 4000 images for training 1000 images for testing on the retrained Inceptionv3 model discussed in 5.2, we obtain the following test confusion matrix: $\begin{bmatrix} 448 & 86 \\ 117 & 349 \end{bmatrix}$. After performing the prediction, we checked manually through several hundred images to determine patterns in correctly predicted images, false negatives, and false positives. Examples of misclassified images are depicted in Figure 5. We hypothesize that patterns can be mistaken as cracks, and vice versa.

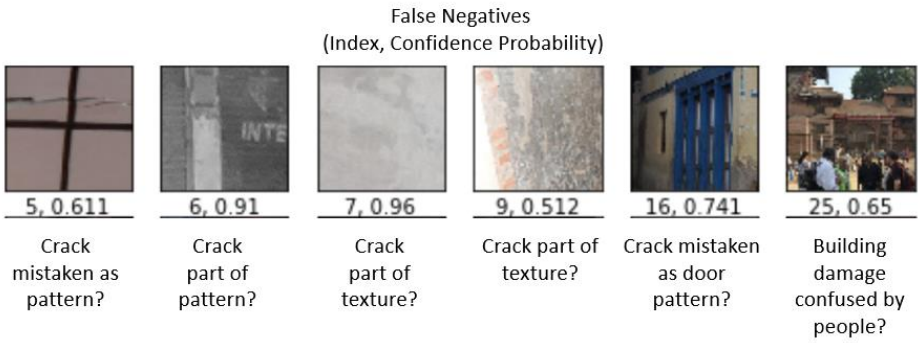

Figure 5: Misclassified images

We augment the data by using horizontal flips, zooms, and shifting of the data. We see that some of these augmentation methods can lead to invalid structural images, as shown in Figure 6.

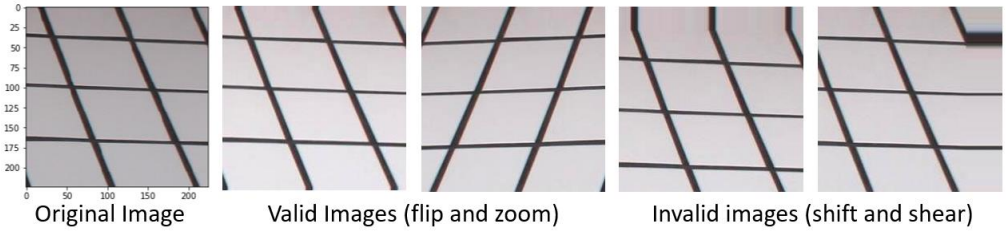

Figure 6: Image Augmentation

5.4 Experiments with Inceptionv3

We retrain the model from 5.2 using Keras, but this time we remove the top layer of the Inceptionv3 network, flatten the output of the penultimate layer, add a fully-connected layer and softmax activation. We find that we are now overfitting (Figure 7, left) most likely because the number of trainable parameters has increased significantly to [2,001,000]. To reduce the variance, we fed in different versions of the training images on each iteration. Since the model never saw the same image more than once, it reduced the overfitting issue (Figure 7, middle). Augmenting with valid images (Figure 7, right) still leads to overfitting, likely because the augmented images are more similar to the original images, but the validation accuracy is also higher, by 1%.

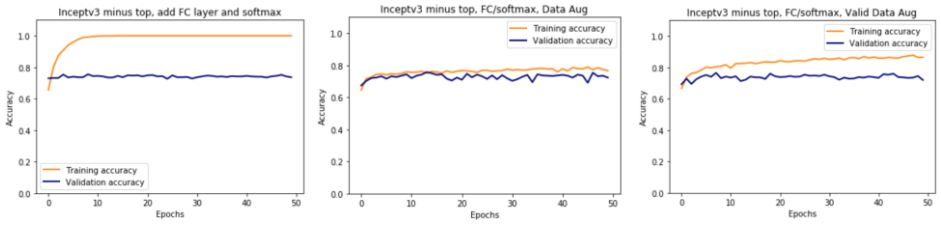

Figure 7: Inceptionv3 minus top layer, no data augmentation (left), data augmentation with shift & flip (middle), data augmentation with flip and zoom (right)

6 Conclusions and Next Steps

We conclude that a variation of a convolutional neural network performs best on our dataset. Furthermore, while bias can be managed by training more parameters (layers) of the CNN, we must be careful not to add so many parameters that we overfit. However, overfitting can be also managed by adding random images to data.

In terms of future work and next steps, more controlled experimentation can be done to manage bias and variance. We could improve validation accuracy by better managing the data (correct mislabeled images, add images similar to the false positives or negatives, cropping irrelevant features, understanding differences in texture or pattern vs. damage, and accommodating wide-angle versus close-up images). Furthermore, other techniques (such as ensemble averaging) could perhaps lead to better performance.

Acknowledgments

We acknowledge Sanjay Govindjee, who alerted us to this problem. The guidance of Fantine Huot and Mark Daoust are also gratefully acknowledged.

References

[1] Pacific Earthquake Engineering Center. 2018. PEER Hub ImageNet Challenge. [ONLINE] Available at: https://apps.peer.berkeley.edu/phichallenge/detection-tasks/. [Accessed 16 October 2018].

[2] Gao, Y. and Mosalam, K. M. (2018), Deep Transfer Learning for Image-Based Structural Damage Recognition. Computer-Aided Civil and Infrastructure Engineering. 2018. [ONLINE] Available at: https://www.researchgate.net/publication/324565121_Deep_Transfer_Learning_for_Image-Based_Structural_Damage_Recognition/. [Accessed 18 October 2018].

[3] Scikit-learn. 2007. Available at: https://scikit-learn.org/ [Accessed 19 November 2018].

[4] TensorFlow for Poets. Available at: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0 [Accessed 19 November 2018].

[5] K-nearest neighbors algorithm. [ONLINE] Available at: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. [Accessed 13 December 2018]

[6] Support vector machine. [ONLINE] Available at: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed 13 December 2018]

[7] Advanced Guide to Inception v3 on Cloud TPU. [ONLINE] Available at: https://cloud.google.com/tpu/docs/inception-v3-advanced. [Accessed 13 December 2018]

[8] Transfer Learning. [ONLINE] Available at: http://cs231n.github.io/transfer-learning. [Accessed 13 December 2018]

[9] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861 [cs], Apr 2017.

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," arXiv:1512.00567 [cs.CV], Dec 2015.

[11] Image Net. [ONLINE] Available at: http://www.image-net.org/. [Accessed 11 November 2018]