Structural Burn Damage Classification with Convolutional Neural Networks

Dillon Pinto

Advisors: Jonathan Ventura, Andrew Fricker

December 15th, 2019

Introduction

The Paradise Camp Fire in Butte County, Northern California, has been the most destructive wildfire in California's history. The fire caused 85 casualties and the destruction of 18,804 structures in the town, spread across over 153,000 acres of land (United States Census Bureau). The largescale damage left $16 billion in insured property losses. With the number of wildfires in California increasing by the year in both quantity and severity, it would be immensely helpful for organization such as Cal Fire to have a tool that can detect structural damage through aerial imagery for post-fire surveying or evacuation purposes. A tool that can accurately identify damaged structures can be extended upon to identify other factors that determine the likelihood of a structure's destruction and therefore help fire protection organizations better mitigate the damage brought on by increasingly recurrent wildfires.

Generating the dataset

The dataset was generated in Google Earth Engine by clipping one rectangular region each for both training and validation sets, of the Paradise Camp Fire aerial imagery. Individual structure images in each set were obtained using the geographic location of each structure's coordinates. A buffer was created around each structure to capture the structure as a whole and any nearby surroundings. This was done since each geographical coordinate represents a "Point" data type and would only encapsulate a pixel of the structure. This meant that a sufficiently large buffer needed to be created around the point to capture the entirety of the structure. Since buildings vary in size, a scale of 30m (length and width) was used to ensure that the majority of structures in the region could be captured. However, this posed the issue of other structures (burnt/unburnt) being included in the buffer, as buildings are often surrounded by other nearby buildings. Despite

the possible issue, it hasn't posed a problem during training and possibly improved classification accuracy.

Using these parameters for image extraction, 412 and 317 images were extracted from the aerial imagery within the training and validation polygons respectively (coordinates shown in Appendix B). Both polygons makeup a small subset of the overall Camp Fire dataset. The training polygon makes up 893,908.8 m^2 in area (0.3451 mi^2), while the validation polygon makes up 354,487.7 m^2 in area (0.13687 mi^2). These are small regions in comparison to the total spread of the Camp Fire - 620,528,776.39 m^2 (239.5875 mi^2). This means much more data can be obtained to better train the network, however heavy bias towards classifying damaged structures is a possible issue due to the Camp Fire leaving few buildings intact. Despite the thousands of remaining datapoints, image extraction proved to be an issue for both storage space requirements and Google Earth Engine's batch processing capabilities, so only datapoints within the training and validation polygons (412 training images, 317 validation images) were used. These images were later augmented to provide a much larger dataset during training.

Extending the dataset

Due to issues faced with batch processing images on Google Earth Engine, the image dataset has been artificially expanded through augmentation with Keras' "ImageDataGenerator" class. This allows the dataset to learn from a multitude of variations in image data including rotations, shifts and flips. The variety in the dataset will expose the network to variations of structure images that wouldn't be available without further image processing on Google Earth Engine and more unique structures for which aerial imagery needs to be extracted. The augmented image data will be used to provide "unique" samples 20,000 times per epoch so the network can adapt to changes in similar imagery.

Architecture

The final architecture for the problem, shown in Appendix A, is a convolutional neural network with 6 convolutional layers and 5 fully-connected layers. The Lambda layer in this network is used to normalize all pixels values being passed into the network. Normalization in the network results in prevention of the exploding gradient problem and allows the network to converge faster.

The emphasis for the convolutional neural network architecture is for it to be narrow and somewhat deep, to a point in which classification accuracy is maximized. The several convolutional layers are used to learn increasingly complex features from the training dataset but are also kept sufficiently narrow as to minimize the possibility of the layers overfitting. Another reason the network was designed to be narrow is to reduce the training time and time taken to tune its hyperparameters for optimal performance. Each convolutional layer is comprised of a 3x3 kernel. This size was chosen to prevent the layers from down-sampling the imagery too much. Since each image is only 123x123 pixels in area, setting the kernel size to anything larger than 3x3 will mean more visual data will be lost during convolution in a low-resolution image that doesn't contain a vast set of features.

The network's fully-connected layers are intended to perform the decision-making task of structure classification, based on the features extracted from the convolutional layers. Theses layers are kept to a maximum width of 200 nodes, in the first layer, and scale-down in size with each subsequent fully-connected layer. A SoftMax activation function is used on the final full-connected layer to produce probabilities that the input maps to each output class.

Optimizing the network's hyperparameters

Learning rate

The learning rate of $5x10^{-5}$ was chosen after trials of learning rate variations between $1x10^{-6}$ to $1x10^{-4}$. The upper-end of the learning rate range caused fluctuation in loss and prevented the network from converging. The lower-end, instead, caused the network to converge very slowly, and wasn't feasible to keep training. The rate of $5x10^{-5}$ was found to result in the quickest convergence.

Number of epochs

The number of epochs was chosen through graphical visualizations of previous training iterations. Extrapolating the graph of the loss function showed that at 20,000 samples per epoch, more than 50 epochs were required to converge. The number of epochs was set to 100 to provide a safe buffer for the loss function to converge. Further training would result in little to no changes in test accuracy.

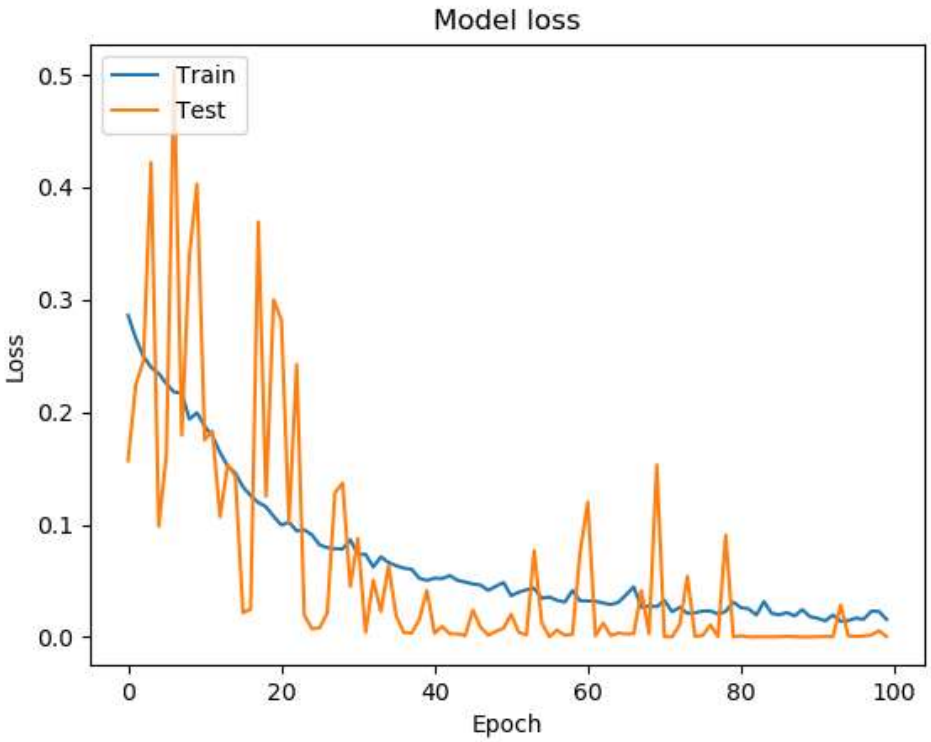

Figure 1: Model loss for the CNN over 100 epochs

Batch size

Since convolutional neural networks are susceptible to changes in batch size, a batch size of 128 samples per batch was chosen as this, empirically, provided the best test accuracy and loss results, while also speeding up the training process (Vanhoucke et al).

Activation function

Every convolutional and fully-connected layer in the network (aside from the output layer), used the ELU activation function. This activation function works well as a solution to the vanishing gradient problem and has been shown to work better when swapped into architectures using ReLU or Leaky ReLU (Clevert et al).

Class Weights

Given the extent of the wildfire, most buildings in paradise were damaged or destroyed, this led to a much smaller sample of undamaged/slightly damaged buildings than damaged buildings. In the dataset used to train the network, the damaged class outnumbers the undamaged class by a 12:1 ratio. To compensate for the imbalance in the dataset, the undamaged class was given a weight 12 times more than the damaged class.

Dropout for regularization

Dropout was applied to reduce the overfitting in the network and provide a better generalization to test data. A dropout of 50% was applied to 3 convolutional layers and the largest fully-dense layer, this provided a vastly better generalization than training without the implementation of dropout.

Results

Given the above hyperparameters for the model described by Appendix A, the results for the validation set were 90.2% accurate. The current threshold to determine if either of the two

classes is more probable is set at 0.5. Figure 2 shows the ROC curve for predictions. The curve shows a modest AUC and while the model is useful in determining structural damage, more work is required to improve the results shown in the ROC curve.

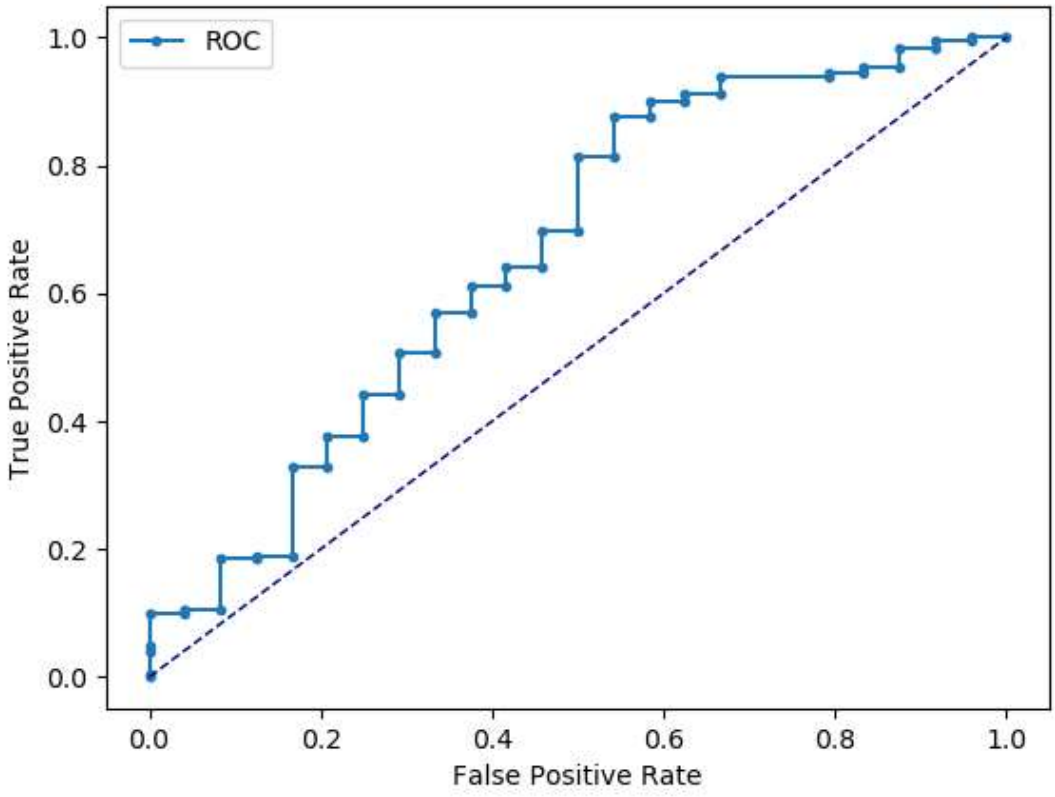

Figure 2: ROC curve for the predictions on the validation set

The ROC curve for the training set predictions is computed as well, to ensure that the model doesn't "memorize" the training set, i.e. overfit and match the training set very closely. The accuracy of the model on the training set is not used as a performance metric, outside of an

overfitting check. The curve provided shows a good trend for the training data but doesn't show sharp overfitting characteristics.

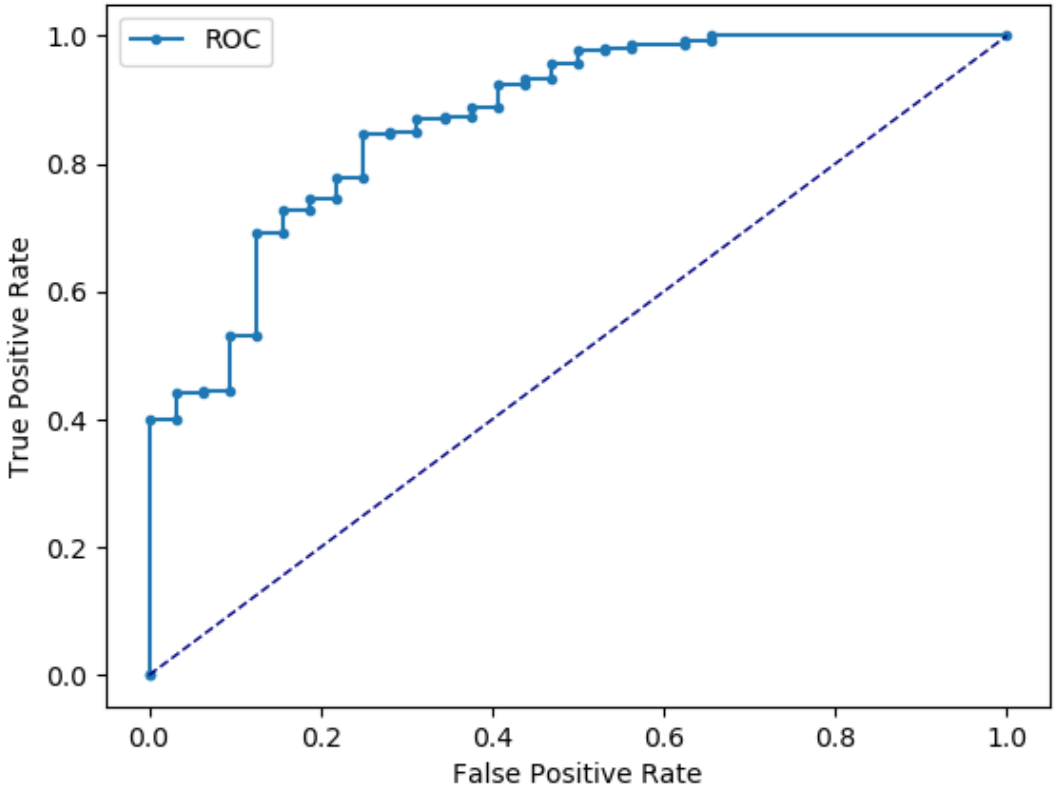

Figure 3: ROC curve for the predictions on the training set

Conclusion

The CNN achieves its goal of determining structural damage through aerial imagery, however improvements need to be made towards further balancing the dataset and choosing an

appropriate threshold to determine if a class is more probable than the other. Despite the class weights added to each class, the model is still biased towards the “damaged” class, due to the sheer number of training examples for that class over the other. This is an improvement in image preprocessing and extraction that needs to be made to further assess the accuracy of the current model.

References

1) "Camp Fire - 2018 California Wildfires." *United States Census Bureau*, United States Census Bureau, www.census.gov/topics/preparedness/events/wildfires/camp.html. Accessed 15 Dec. 2019.

2) Clevert, Djork-Arne, et al. "FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS (ELUS)." *ICLR*, 22 Feb. 2016.

3) Vanhoucke, Vincent, et al. "Improving the speed of neural networks on CPUs." *Google Research*, 2011.

Appendices

Appendix A – Keras CNN model

```
model.add(Lambda(lambda x: x / 127.5 - 1, input_shape=input_shape))
model.add(Conv2D(64,(3,3),activation="elu",strides=(2,2),
data_format='channels_last'))
model.add(Conv2D(64, (3, 3), activation="elu", strides=(2, 2)))
model.add(Dropout(keep_prob))
model.add(Conv2D(64, (3, 3), activation="elu"))
model.add(Conv2D(64, (3, 3), activation="elu"))
model.add(Dropout(keep_prob))
model.add(Conv2D(64, (3, 3), activation="elu"))
model.add(Conv2D(64, (3, 3), activation="elu"))
model.add(Dropout(keep_prob))

model.add(Flatten())
model.add(Dense(200, activation="elu", use_bias=True))
model.add(Dropout(keep_prob))
model.add(Dense(100, activation="elu", use_bias=True))
model.add(Dense(100, activation="elu", use_bias=True))
model.add(Dense(10, activation="elu", use_bias=True))
model.add(Dense(2, activation="softmax"))
```

Appendix B – Polygon coordinates

I. Training Polygon
[-121.61112971473352,39.761907941249994]
[-121.5891356008114,39.761907941249994]
[-121.5891356008114,39.76617993704935]
[-121.61112971473352,39.76617993704935]
[-121.61112971473352,39.761907941249994]

II. Validation Polygon
[-121.59506929487134,39.78155804409371]
[-121.58609998792554,39.78155804409371]
[-121.58609998792554,39.78571339763988]
[-121.59506929487134,39.78571339763988]
[-121.59506929487134,39.78155804409371]