Ayina Anyachebelu
MUSA 650
May 2020
Final Project

Classification of Forest Fires in Arizona Using Drone Imagery

1. Introduction

After previously exploring the impact of the 2019-2020 bushfires on Australia's Kangaroo Island in the course, I have been interested in the rise of unprecedented wildfires globally. Climate change has resulted in increased heat, unpredictable rain and snow patterns and changes in plant communities that have made it more likely for fires to begin and burn more severely (Borunda, 2020). These fires are not only responsible for the immense loss of human lives and property but also decimate our ecosystems and hinder tree regeneration, leading to more reburns and a vicious cycle (Halofsky et. Al, 2020). As global warming worsens, it is imperative that fire scientists, resource managers, and the general public are able to identify and tackle forest fires more effectively. Improved fire management and scheduling for future incidents as well as lower life risks for firefighters depend on the ability to predict fire behavior. Recent research demonstrates that aerial imagery has much potential in the field of wildfire exploration (Shamsoshoara, 2020). The goal of this project is to classify aerial imagery of forests based on if they depict a fire or not using machine learning and deep learning.

2. Data

The data comes from "The FLAME Dataset: Aerial Imagery Pile Burn Detection Using Drones (UAVS)" from the IEEE Data portal (Link). It contains images during a prescribed pile burn in Arizona. The training and valuation set consists of 39,375 images that were resized to 254x254. Of these images, 25,018 of them are labelled as fire images and 14,357 are labelled as non-fire. With regards to the test data, it consists of 8617 labelled images, 5137 of which were fire and 3480 were non-fire. All images are in JPEG format and have the three RGB channels.

3. Methods

3.1 Overview

Part 1 of the project involves an attempt to build an autoencoder to be used for dimensionality reduction given the research demonstrating that autoencoders can have smaller errors than Principal Component Analysis (PCA) (Hinton et. al, 2006). Part 2 includes an attempt to use the encoded images to fit the best Support Vector Machine (SVM) model indicated by a grid search. Part 3 of the project involves creating and testing a custom convolutional neural network (CNN). To evaluate the performance of the algorithms I used confusion matrices, accuracy, f-1 scores, and Precision-Recall curves. I used the f-1 score because my dataset was imbalanced, and f-1 takes into consideration precision and recall as well as the distribution of the data. I especially focused on the recall of the fire class as well as the Precision-recall curve because I am interested in the positive class considering that detecting a forest fire is important and false negatives would be costly in this situation. The PR AUC uses the positive predictive value PPV and true positive rate TPR, so it is more sensitive to the improvements for the positive class (Davis et. al).

3.2 Autoencoder

The goal of this method was to build a convolutional autoencoder that would compress the input data into a latent-space representation which would be used to reconstruct the original images through the decoder subnetwork. Because of 254 x 254 is not a multiple of 4, I added a zero padding layer at the top of my network to create an output shape of 256X256 and ended with a Cropping2D layer to return to an output size consistent with my input.

With regards to the decisions I made when creating the autoencoder, I chose convolutional layers to detect patterns with 3x3 filters. The 3x3 filter appears to be the standard since having an odd side length of the filter covers the pixel of interest and its neighbors from all sides and 3x3 requires the least processing time. With regards to the strides, I found that a stride of 2 is typical on two dimensional feature maps are also standard. Additionally, learning rates are often a small positive value between 0 and 1 with 0.1 being a traditionally common value. I initially decided to use 0.01 to allow the autoencoder model to learn a more optimal or hopefully even a globally optimal set of weights. Finally, I used the "mse" loss function because I found this was common across autoencoders. However, with these parameters, my autoencoder reconstructed completely blank images and the network's validation loss never decreased. After researching this issue, I discovered that possible problems included a "dying relu" problem or a learning rate that was too high. The "dying relu" problem results from the fact that the input for an activation can be negative, due to operation performed in the network. Using the "LeakyRelu" activation instead ensures the gradient descent has a non-zero value and the network continues learning without reaching dead end. After making these changes, the autoencoder was able to reconstruct the images with a latent space of 32 X 32 X 32, reducing the dimensions from 193,548 to 32,768.

```
Model: "Auto-Encoder_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d (ZeroPadding2 (None, 256, 256, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 256, 256, 8)       224
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 8)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 16)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 16)        528
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 64, 64, 16)        0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 128, 128, 8)       136
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 128, 128, 8)       0
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 256, 256, 8)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 256, 256, 3)       27
_________________________________________________________________
cropping2d (Cropping2D)      (None, 254, 254, 3)       0
=================================================================
Total params: 6,723
Trainable params: 6,723
Non-trainable params: 0
```

Summary of Autoencoder

3.3 SVM

I began with an SVM model using the encoded image set with the reduced number of dimensions. I ran the autoencoder using only the encoder part of the model and saved these images and normalized them. Using, 15% of the original dataset, the SVM model parameters were selected based on PR AUC values using GridsearchCV from the following combinations: kernel was selected from 'rbf' and 'linear', C was selected from powers of 10 from 10^{-3} to 10^{3} and gamma was selected from 10^{-5} to 10^{-1}. The GridsearchCV was conducted with nested cross validation of 5 folds in the inner and outer loop. The combination that performed the best consistently was 'rbf', C=10.0, and gamma=0.0001. These parameters were then used to create an SVM model to fit the whole dataset.

3.4 CNN

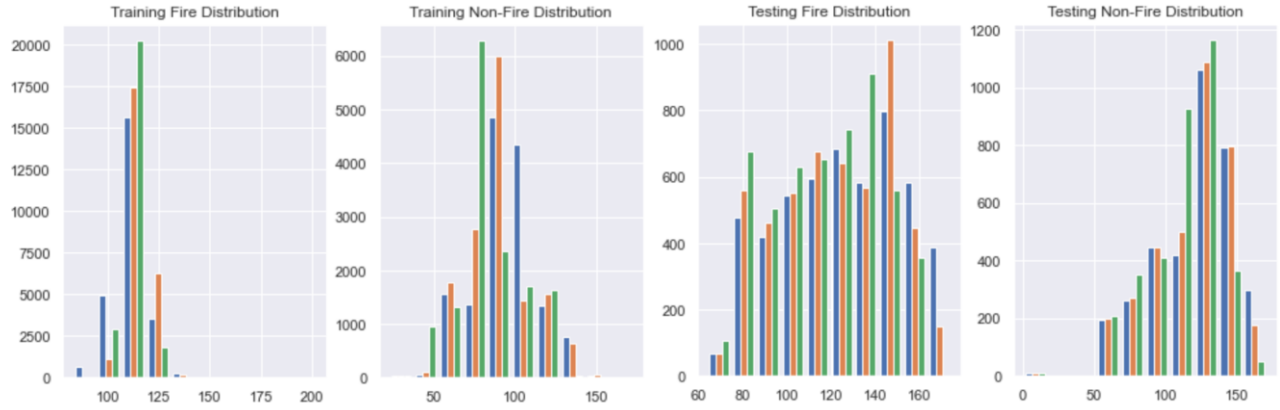
To build my neural network, I used three sets of convolutional layers followed by max pooling layers for feature extraction. I chose the number of feature maps based on readings showing that common practice is using powers of 2 especially beginning at 32 for the number feature maps, the number of neurons in a dense layer and the batch size (Goodfellow, 276). Using 15% of the data, I conducted a GridsearchCV with 5-fold cross validation in order to choose the number of neurons in the hidden layer (between 64 and 128) and to choose the batch size (between 32 and 64) to find out which parameters would give me the best results over 5 epochs. The best combination was 64 neurons and a batch size of 32. A dropout of 0.5 was chosen based on articles indicating that starting from a dropout from 0.5 and up to 0.8 was ideal to avoid overfitting. I chose my learning rate of 0.0001 based on the trial and error I had undergone while creating the autoencoder and realizing this worked for 15% of the dataset. I also created a custom metric using the average precision score which is a proxy for the PR AUC from sklearn that I used for my model compilation.

After parameter tuning, I loaded the entire dataset using the Image Data Generator, including some data augmentation by randomly zooming images or horizontally flipping them. Instead of tuning the number of epochs, I used the early stopping callback.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 252, 252, 32)      896
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 252, 252, 32)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 126, 126, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 124, 124, 64)      18496
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 124, 124, 64)      0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 60, 60, 64)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                3686464
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 3,742,849
Trainable params: 3,742,849
Non-trainable params: 0
```

Summary of CNN

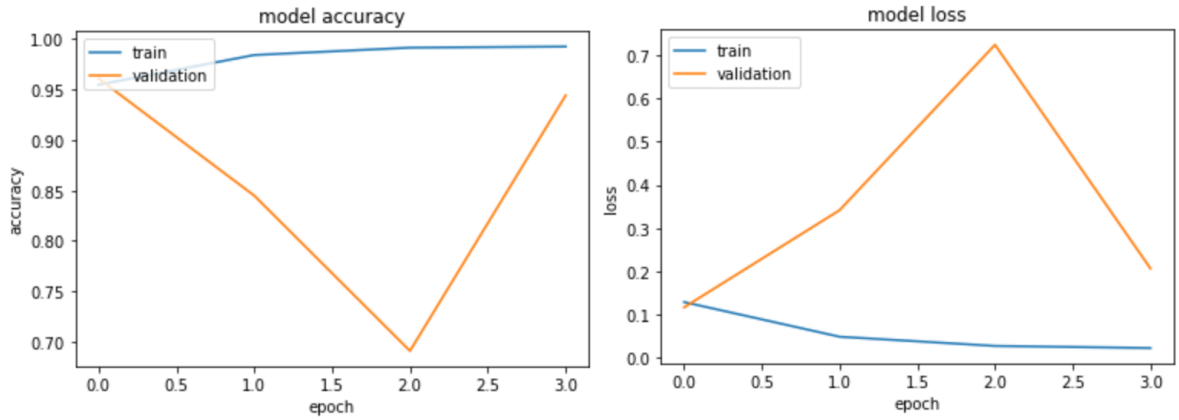
After the first results of the CNN on the test data were poor (explained in the Results section), I attempted different methods to reduce overfitting. The techniques I tried included increasing the amount of data augmentation to include rotations, adding a batch normalization layer after every convolutional layer and increasing the amount of drop out from 0.5 to 0.7 and removing one of the convolutional /max pooling layers combinations. I checked the success of these by using the test data as the validation data. In all instances, the training loss decreased, and training accuracy was very high while it was poor for validation. This meant that the data was still overfitting. Consequently, I suspected that the training data must have a fairly different distribution from the test data given the model's inability to translate what it was learning from the training set. The distributions of the colors for the three channels for the training and test sets below supported my suspicions. The charts show that the distributions of the average pixel value for each photo is quite different between the training and testing sets.



I aggregated the training and data set and created a custom split of 0.25 for validation and used this set to retrain the CNN (the version that had been altered to avoid overfitting). With this custom split the validation loss was able to decrease significantly the more the model learned and reach an accuracy of 0.96 after the 5th epoch. From these findings, I concluded it was best to aggregate the sets and make custom splits of the training and test sets. I moved about 25% of the fire and non-fire images from the aggregated set to create a new test set which was put aside. The rest of the data was split into the training and validation sets. The new sets' distribution of images were 28,789 training images, 7196 validation images and 11998 testing images. Below is the final convolutional neural network I used which was an altered version of the first one to reduce overfitting and remove unnecessary layers. When the model was trained with this new split of the data set, within the first three epochs the validation loss was as low as 0.116 and the best validation accuracy was 0.96 as seen below in the charts.

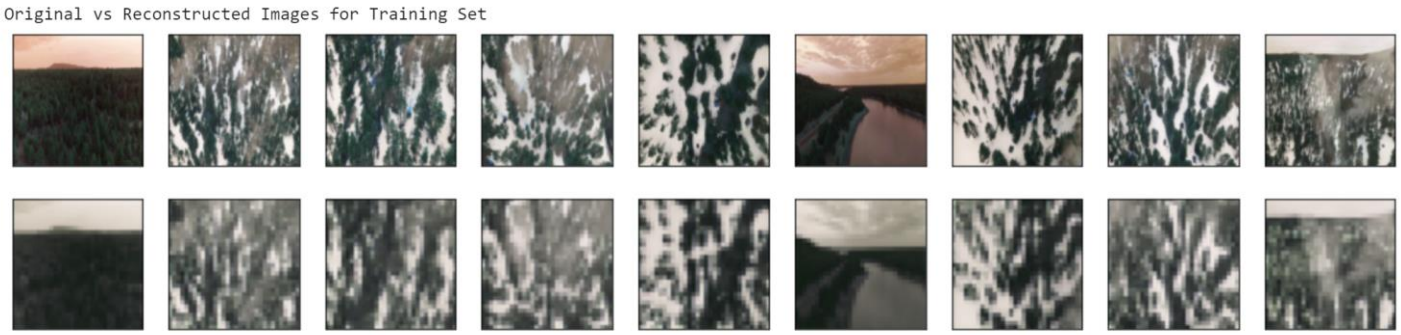
```
Model: "Final_CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 254, 254, 32)      128
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 254, 254, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 127, 127, 64)      18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 127, 127, 64)      256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 127, 127, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 254016)            0
_________________________________________________________________
dense (Dense)                (None, 64)                16257088
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation (Activation)      (None, 1)                 0
=================================================================
Total params: 16,277,185
Trainable params: 16,276,865
Non-trainable params: 320
```

Summary of Final CNN

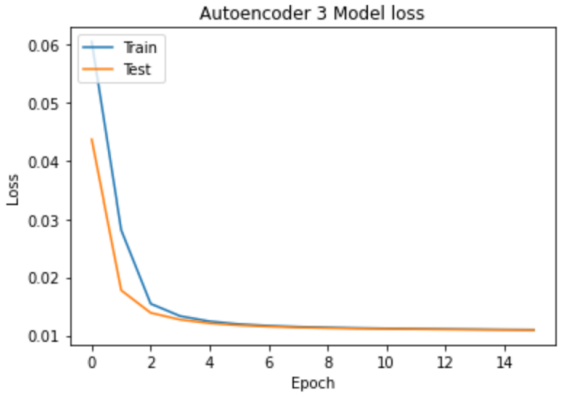


4. Results

4.1 Autoencoder Results



The autoencoder was able to produce a low loss of 0.0109 after 15 epochs. It was also able to reconstruct the images fairly well as seen above. However, it did not extract the little fires seen in the third and fourth original photos above. It appeared that the autoencoder was identifying these little fires as noise and failing to extract them when it reconstructed the images. As a result, it was likely that the autoencoder may not have been appropriate for this specific problem because it would remove fires from some photos that were labelled as fire. Additionally, the fact that the autoencoder was already failing to extract some fires with a latent space as large as 32 X 32 X 32 signified that adding extra layers to the autoencoder to further reduce the number of features of the images would not result in good reconstructions.



*Note: The “test” in Autoencoder loss graph is from a validation split of 0.2 of the training data

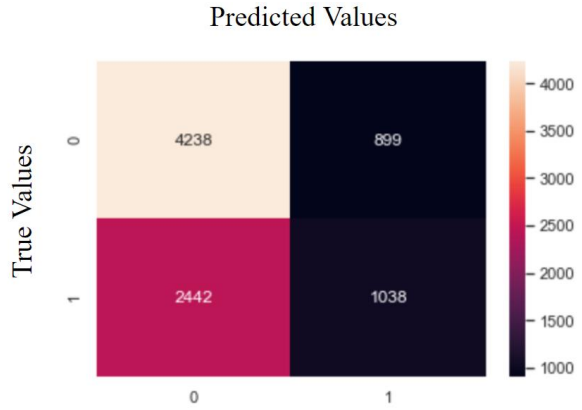
4.2 SVM Results

A grid search for parameter tuning of the SVM model was conducted using 15% of the data with dimensions that had been reduced by the autoencoder. Despite doubts about the functionality of the autoencoder, the 5-fold nested cross-validated PR AUC for the grid search was 0.986. Since this was conducted using a small part of the dataset, it is possible that the set did not include many of the images with the small fires.

Subsequently, I attempted to run the best SVM model on all the encoded images. However, given that the autoencoder still retained over 32,000 features of the images, the model ended up training for over 70 hours without converging. Consequently, the results were inconclusive.

4.3 CNN Results

The results of the CNN on the test set were quite negative. Using a threshold of 0.5, the accuracy was 0.61 and the weighted average f-1 score was 0.58. However, the recall for the fire class was 0.82, which was good as the model was not producing many false negatives because in the case of forest fires false negatives are very costly. Nevertheless, the model classified many of the "non-fire" images (2442 of 3480 images) as fire as seen in the confusion matrix below. While the model had performed well on the training and validation data, the results were very poor on the testing data.

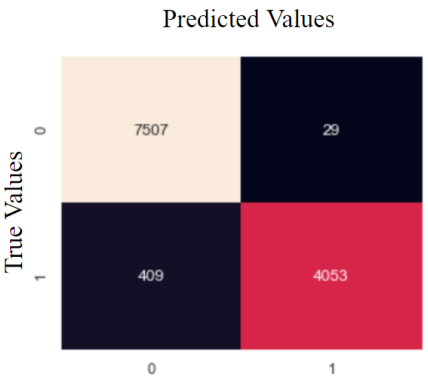


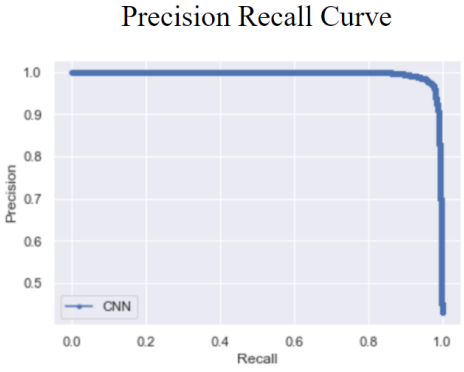
	precision	recall	f1-score
0	0.63	0.82	0.72
1	0.54	0.30	0.38
accuracy			0.61
macro avg	0.59	0.56	0.55
weighted avg	0.59	0.61	0.58

**Note: The fire class is 0 and the non-fire class is 1

As mentioned in the Methods sections, the CNN was altered to account for overfitting and new training, validation, and testing sets were created from a custom split of an aggregation of all the original sets. With this new split of the dataset, the model performed extremely well. The testing accuracy was 0.96 and the area under the precision-recall curve (shown below) was 0.995. The recall for the fire class was almost 100% with only 29 of the 7536 fire images being classified as non-fire. Before creating the custom split, a major problem in the performance of the model was the significant number of false positives, however, the non-fire class had high precision and recall.

	precision	recall	f1-score
0	0.95	1.00	0.97
1	0.99	0.91	0.95
accuracy			0.96
macro avg	0.97	0.95	0.96
weighted avg	0.96	0.96	0.96





5. Discussion

Although I had attempted to build the autoencoder for dimensionality reduction, the fact that it tended to not extract the small fires from images revealed that while using an autoencoder may be helpful in many scenarios, it was not an appropriate approach to solve the specific problem I was looking at. Moreover, though the autoencoder reconstructed the images fairly well when it reduced the dimensions from 254 x 254 x 3 to a latent space of 32 x 32 x 32, it still had 32,768 features. In that case, not only was the autoencoder not extracting many fires, it was doing this while retaining many features which defeated its purpose of dimension reduction. This led into the issue of the SVM model's inability to conclusively fit the dataset within a decent amount of time. Considering, the parameters of the SVM model as well as the tens of thousands of features it had to use even after encoding, it was not possible for the SVM to converge within a reasonable amount of time. Due to the time constraints, I had to move on after the model had been training for several days and had repeatedly resulted in dying kernels, making my results from that model inconclusive.

The dataset I had ultimately had the issue of the images in the training set being quite different from those in the testing set which is why regardless of how much was learned by the convolutional neural network, this could not be translated to good testing results. It was best for me to follow my suspicions about the difference in the distributions of the images and manually aggregate, randomize, and split the images in the training and testing sets because this allowed me to see what the actual performance of the neural network was and resulted in the positive results at the end.

Before I realized the main issue with the difference in the images of the training and test sets, I thought the only problem with my model was overfitting and I experimented with trying to reduce it. This ended up being a blessing in disguise because it forced me to try different methods, some of which I incorporated into the final CNN with the new data split. For example, I initially did not have any batch normalization layers and only one dropout layer near the end of the architecture, but I ultimately added a batch normalization layer after every convolutional/max pooling layer set. I also experimented with reducing the number of sets of convolutional/max pooling layers from 3 to 2. While this did not change the overfitting issue, the performance on training and validation did not change which revealed that the third set of layers was unnecessary, teaching a lesson that sometimes simpler models can be just as good as complex ones.

Going further, it would be interesting to see how the CNN would perform on identifying fires on forests outside of Arizona and what alterations can be made to further increase its generalization so that what it learns from this dataset can be transferred well to forest fire management problems in other locations.

References

Borunda, A. (2020). The science connecting wildfires to climate change. National Geographic. Published 17 September 2020. https://www.nationalgeographic.com/science/article/climate-change-increases-risk-fires-western-us

Davis, J. & Goadrich, M. The Relationship Between Precision-Recall and ROC Curves. University of Wisconsin-Madison. https://www.biostat.wisc.edu/~page/rocpr.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning: Adaptive Computation and Machine Learning Series. Massachusetts Institute of Technology. Published 2016. https://www.deeplearningbook.org/contents/optimization.html

Halofsky, J.E., Peterson, D.L. & Harvey, B.J. (2020). Changing wildfire, changing forests: the effects of climate change on fire regimes and vegetation in the Pacific Northwest, USA. fire ecol 16, 4. https://doi.org/10.1186/s42408-019-0062-8

Shamsoshoaraa, A., Fatemeh, A., Abolfazl, R., Zheng, L., Fule, P. & Blasc, E. (2020). Aerial Imagery Pile burn detection using Deep Learning: the FLAME dataset. Northern Arizona University. Published 28 December 2020. https://arxiv.org/pdf/2012.14036.pdf

Hinton, G. & Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. Science Magazine, 313, 504-507. Published 28 July 2006. https://www.cs.toronto.edu/~hinton/science.pdf