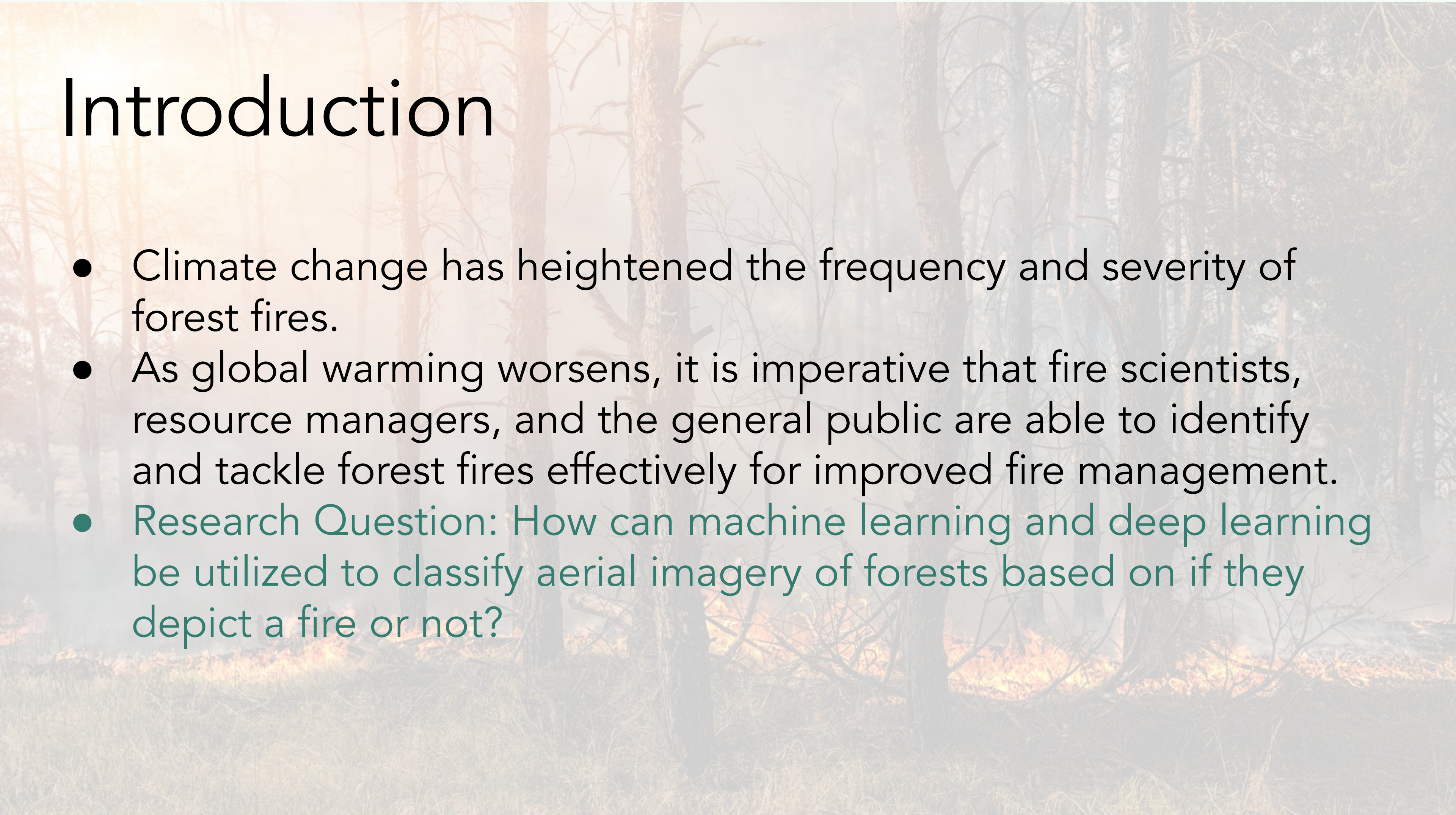
Detecting Forest Fires Using Drone Imagery

MUSA 650: Machine Learning for Remote Sensing

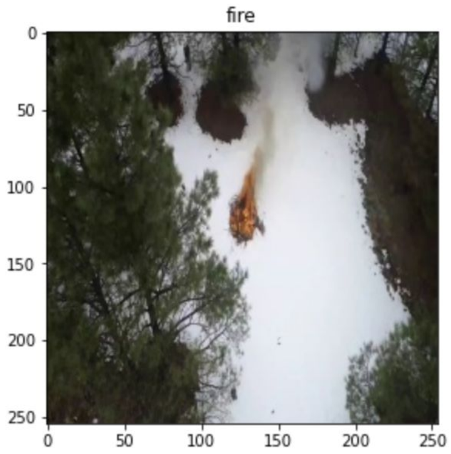
Ayina Anyachebelu (W’22, C’22)

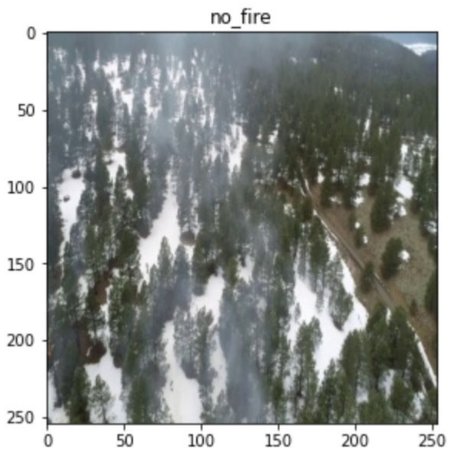


Introduction

- Climate change has heightened the frequency and severity of forest fires.
- As global warming worsens, it is imperative that fire scientists, resource managers, and the general public are able to identify and tackle forest fires effectively for improved fire management.
- Research Question: How can machine learning and deep learning be utilized to classify aerial imagery of forests based on if they depict a fire or not?

The FLAME Dataset: Aerial Imagery Pile Burn Detection Using Drones





- IEEE Data Portal dataset that contains images from a pile burn in an Arizona forest
- Images of size 254 X 254 with RGB channels
- Training set: 39375 images
  - 25018 images of fire (63%)
  - 14357 images of non-Fire (37%)
- Test Set: 8617 images
  - 5137 images of fire (59%)
  - 3480 images of non-fire (41%)

Methods

1 Autoencoder

2 Support Vector Machines

3 Convolutional Neural Network

Metrics

- Confusion matrices and accuracy, f-1 score (because of imbalanced dataset)
- A focus on PR AUC (which uses the positive predictive value PPV and true positive rate TPR) because of the specific interest in the positive (fire) class given that false negatives are costly in the case of forest fires

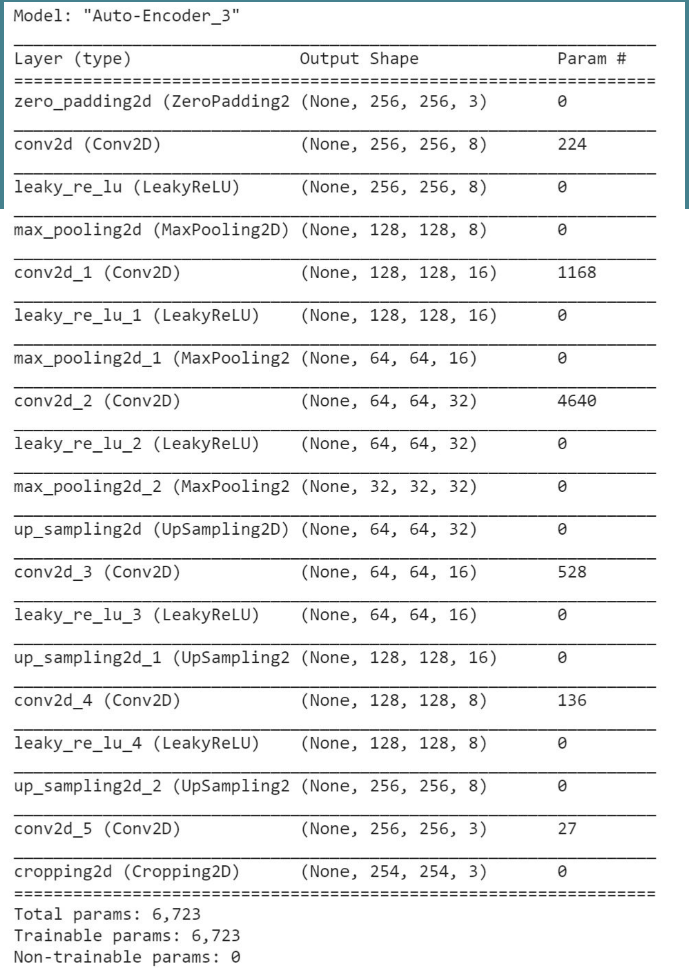
Autoencoder

Purpose

- Dimension reduction with smaller errors than PCA
- Compress the input data into a latent-space representation which would be used to reconstruct the original images with less features

Parameter Decisions

- 3X3 filters and strides of 2 are standard in this field
- Initial learning rate of 0.01 reconstructed blank images so I reduced learning rate to 0.0001 with Adam optimizer
- Employed the “LeakyRelu” activation to avoid “dying Relu” problem
- Autoencoder reduced dataset to a latent space of 32 X 32 X 32



```
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
```

Support Vector Machine

1

- Using 15% of the dataset with reduced dimensions using the autoencoder, SVM parameters were selected based on PR AUC
- GridsearchCV Nested Cross Validation with 5 folds choices:
  - Kernel: “rbf” and “linear”
  - Gamma: powers of 10 from 10^{-5} to 10^{-1}
  - C: powers of 10 from 10^{-3} to 10^{3}
- Best Combination: “rbf”, gamma = 0.0001, C=10.0

2

- The entire data set’s dimensions were reduced using encoder and saved
- These parameters were then used to create an SVM model to fit the whole dataset.

Convolutional Neural Network

Parameter Decisions

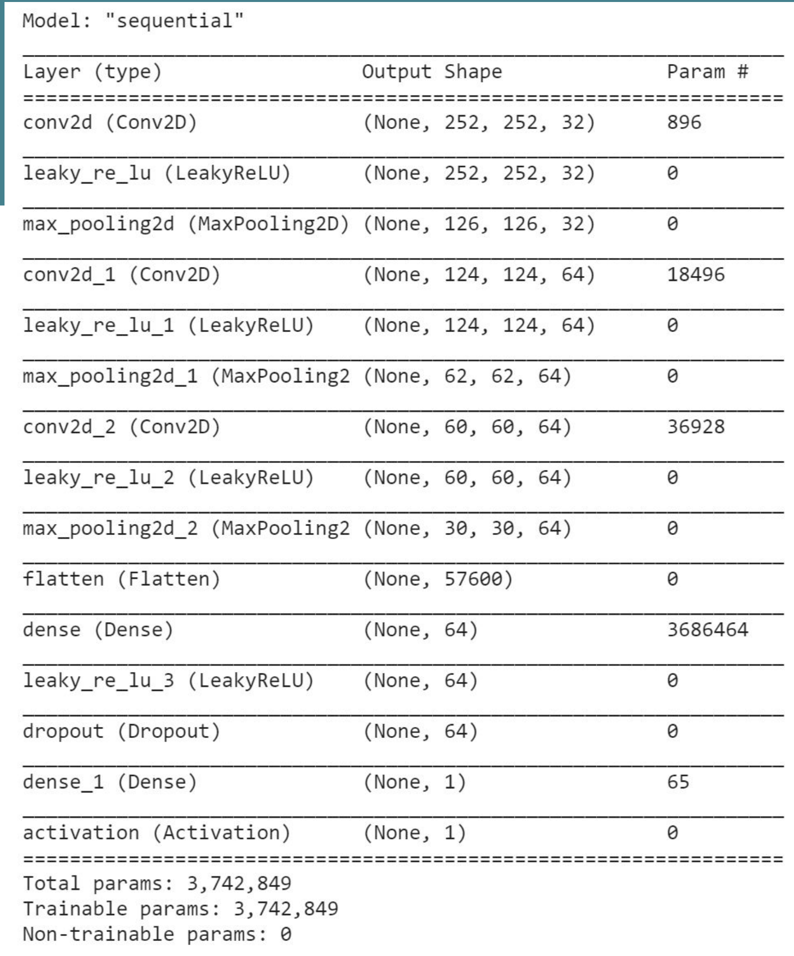
- Number of feature maps chosen based on readings showing that powers of 2 starting from 32 is standard (same for batch size)
- Dropout of 0.5 based on articles indicating 0.5 to 0.8 is ideal to avoid overfitting
- Learning rate of 0.001 chosen based on trial and error from training autoencoder

Parameter Tuning

- GridsearchCV with 5 folds choices:
  - # neurons in dense layer: 64 vs 128
  - Batch Size: 32 vs 64
- Best Combination: 64 and 32

Processes

- Created a customer metric based on PR AUC from sklearn
- Utilized the Keras Image Data Generator including data augmentation (zoom, horizontal flip) to load in all images



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 252, 252, 32)      896
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 252, 252, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 126, 126, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 124, 124, 64)      18496
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 124, 124, 64)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 60, 60, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 57600)             0
_________________________________________________________________
dense (Dense)                (None, 64)                3686464
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
_________________________________________________________________
activation (Activation)      (None, 1)                 0
=================================================================
Total params: 3,742,849
Trainable params: 3,742,849
Non-trainable params: 0
```

CNN: Reducing Overfitting

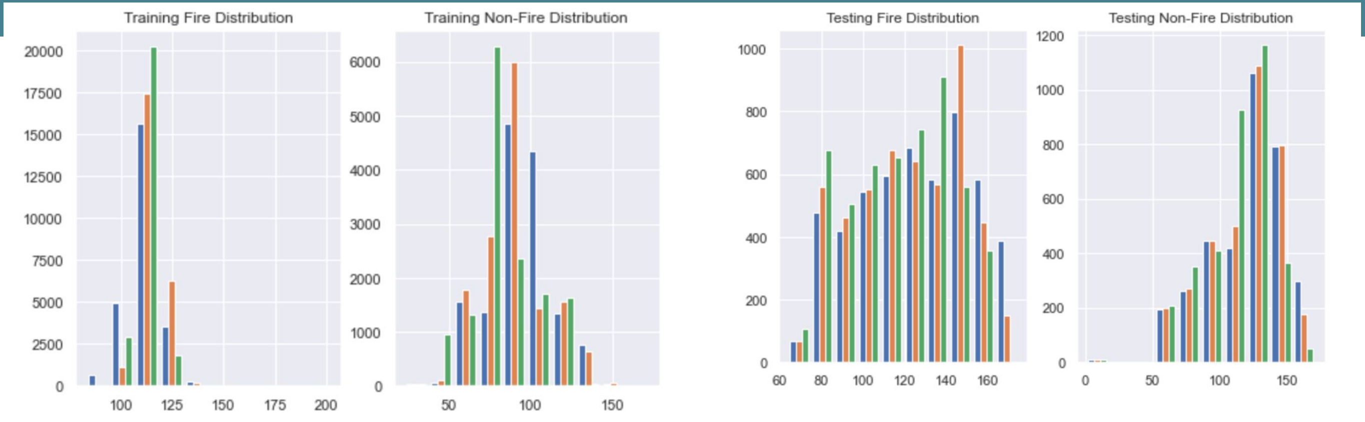
1 — 2 — 3 — 4

1. Increasing amount of data augmentation (zooming, flips, rotations)
2. Including Batch Normalization after every convolutional layer
3. Increasing the amount of dropout after my dense layer
4. Removing one of my convolutional/max pooling layer sets

Results

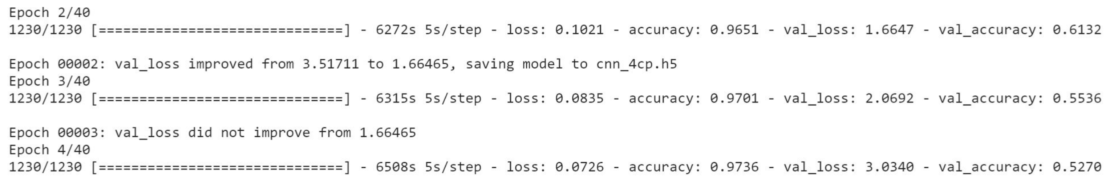
- None of these trials actually resulted in better generalization and improved performance on the testing data
- However, the fact that removing one of my convolutional/max pooling layers sets had similar model performance revealed that I did not need those extra layers

CNN: Inspecting the distributions of the dataset

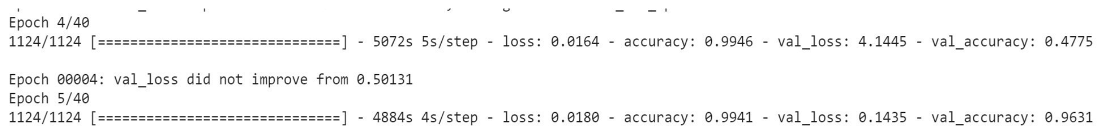


I suspected that the failure of my model to transfer what it learned to predict the test set was due to inherently different sets in the training vs test images. Plotting the distributions of the channels (shown left) from the sets seemed to support my theory.

Additionally I put my dataset as the validation set and retrained my CNN and the fact that the training loss consistently went down with increasingly worse performance on the validation set (top image). However, when I aggregated the training and test sets and did a retrained the models with a validation split, validation accuracy was able to reach 0.96 which further supported my theory (bottom image).
Going forwarded, I created a custom split to tackle this.



```
Epoch 2/40
1230/1230 [==============================] - 6272s 5s/step - loss: 0.1021 - accuracy: 0.9651 - val_loss: 1.6647 - val_accuracy: 0.6132

Epoch 00002: val_loss improved from 3.51711 to 1.66465, saving model to cnn_4cp.h5
Epoch 3/40
1230/1230 [==============================] - 6315s 5s/step - loss: 0.0835 - accuracy: 0.9701 - val_loss: 2.0692 - val_accuracy: 0.5536

Epoch 00003: val_loss did not improve from 1.66465
Epoch 4/40
1230/1230 [==============================] - 6508s 5s/step - loss: 0.0726 - accuracy: 0.9736 - val_loss: 3.0340 - val_accuracy: 0.5270
```



```
Epoch 4/40
1124/1124 [==============================] - 5072s 5s/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 4.1445 - val_accuracy: 0.4775

Epoch 00004: val_loss did not improve from 0.50131
Epoch 5/40
1124/1124 [==============================] - 4884s 4s/step - loss: 0.0180 - accuracy: 0.9941 - val_loss: 0.1435 - val_accuracy: 0.9631
```

Data | Methods | Results | Discussion

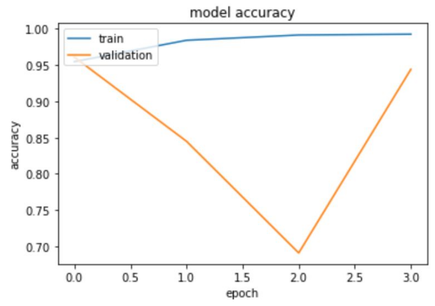
CNN: Retraining

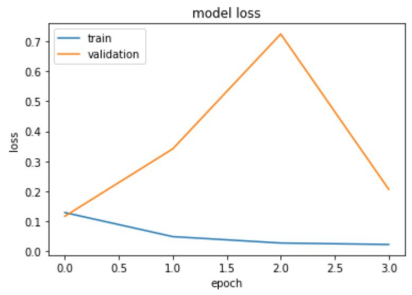
New Custom Data Split

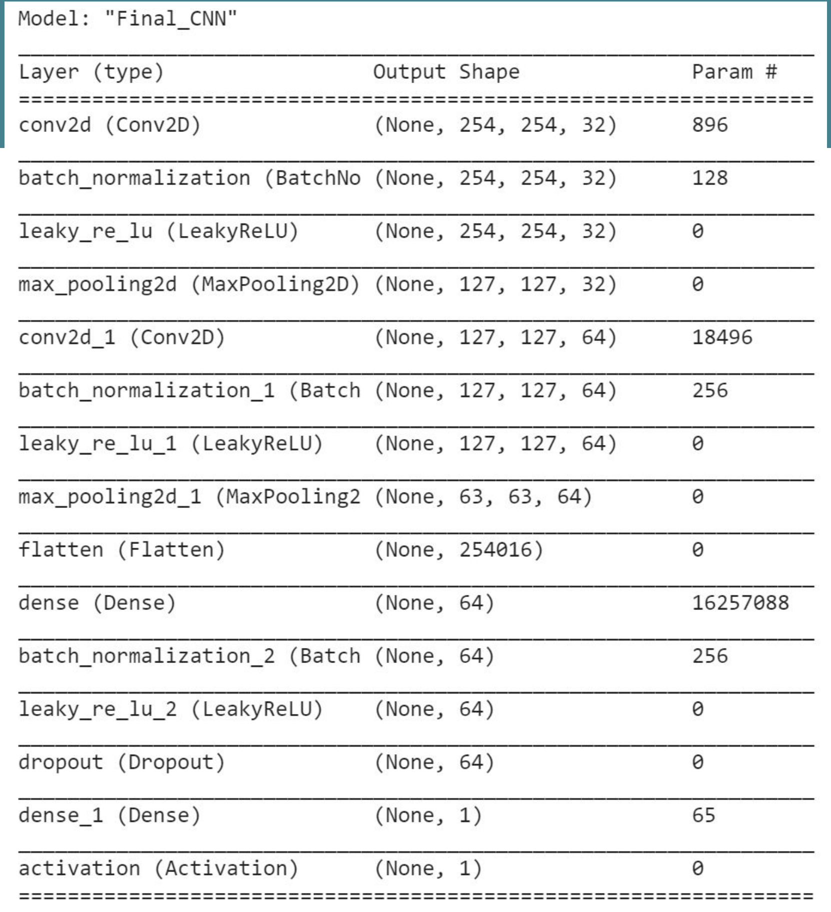
- I manually aggregated, randomized and split the data into training and testing & distribution of classes was similar to original
- New Training Set: 28789 images and 7196 for validation
- New Testing Set: 11998 images

Retraining altered CNN on new set

- A final CNN (shown right) was created taking into account the need to reduce overfitting from my previous trials
- Within the first 3 epochs validation loss was as low as 0.116 and validation accuracy was as high as 0.96







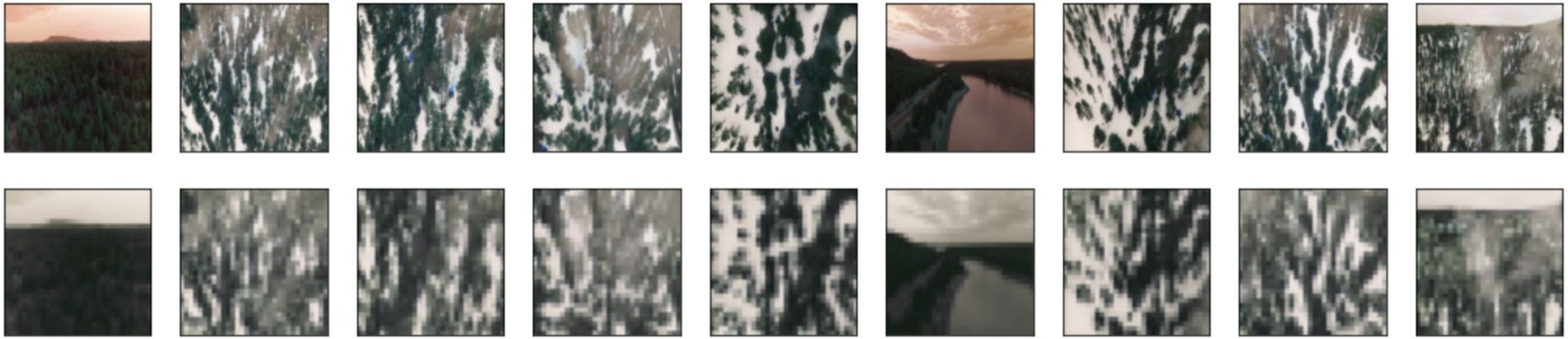
```
Model: "Final_CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
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
=================================================================
```

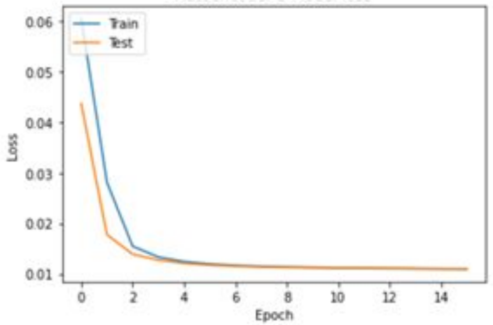
Results

1 Autoencoder — 2 Support Vector Machine — 3 Convolutional Neural Network

Autoencoder

Row 1: Original Images
Row 2: Reconstructed Images





- The autoencoder was able to produce a low loss of 0.0109 after 15 epochs and reconstructed images fairly well as seen above
- It did not extract the little fires seen in the third and fourth original photos above. It appeared that autoencoder was identifying these little fires as noise and failing to extract these features
- It seemed 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

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

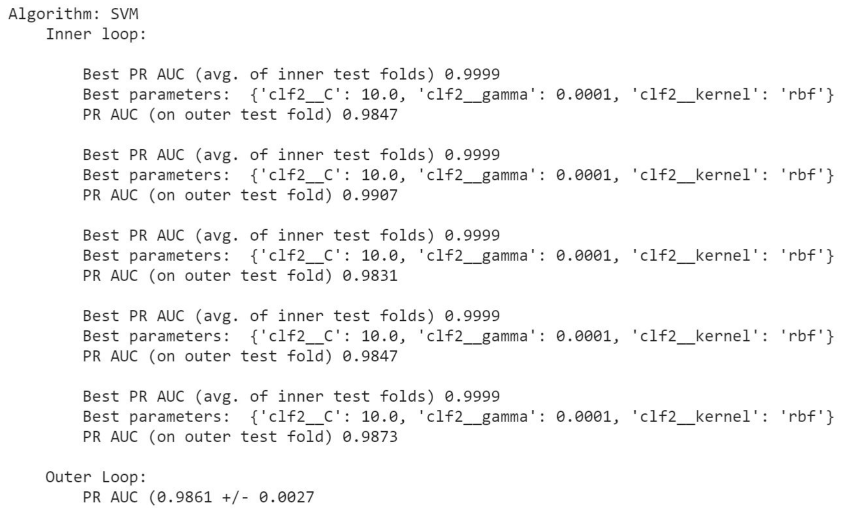
Support Vector Machine

Results of Grid Search (in photo to the right)

- 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 only 15% of the dataset, it is possible that the set did not include many of the images with the small fires.

Results of Training and Testing Best SVM Model

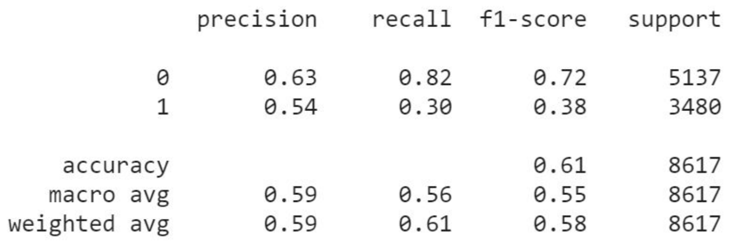
- I attempted to run the best SVM model on all images but given that the autoencoder still retained over 32,000 features of the images, the model ended up training for over 99 hours without converging.
- Since the autoencoder was already failing to extract some fires with its large latent space adding extra layers to further reduce the number of features of the images would not result in good reconstructions.



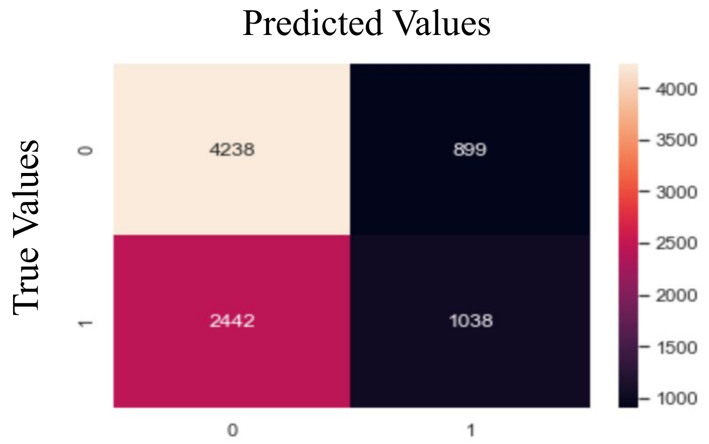
```
Algorithm: SVM
    Inner loop:

        Best PR AUC (avg. of inner test folds) 0.9999
        Best parameters:  {'clf2__C': 10.0, 'clf2__gamma': 0.0001, 'clf2__kernel': 'rbf'}
        PR AUC (on outer test fold) 0.9847

        Best PR AUC (avg. of inner test folds) 0.9999
        Best parameters:  {'clf2__C': 10.0, 'clf2__gamma': 0.0001, 'clf2__kernel': 'rbf'}
        PR AUC (on outer test fold) 0.9907

        Best PR AUC (avg. of inner test folds) 0.9999
        Best parameters:  {'clf2__C': 10.0, 'clf2__gamma': 0.0001, 'clf2__kernel': 'rbf'}
        PR AUC (on outer test fold) 0.9831

        Best PR AUC (avg. of inner test folds) 0.9999
        Best parameters:  {'clf2__C': 10.0, 'clf2__gamma': 0.0001, 'clf2__kernel': 'rbf'}
        PR AUC (on outer test fold) 0.9847

        Best PR AUC (avg. of inner test folds) 0.9999
        Best parameters:  {'clf2__C': 10.0, 'clf2__gamma': 0.0001, 'clf2__kernel': 'rbf'}
        PR AUC (on outer test fold) 0.9873

    Outer Loop:
        PR AUC (0.9861 +/- 0.0027
```

Altogether, given time constraints, the results of the SVM were inconclusive.

Convolutional Neural Network (Initial Training)



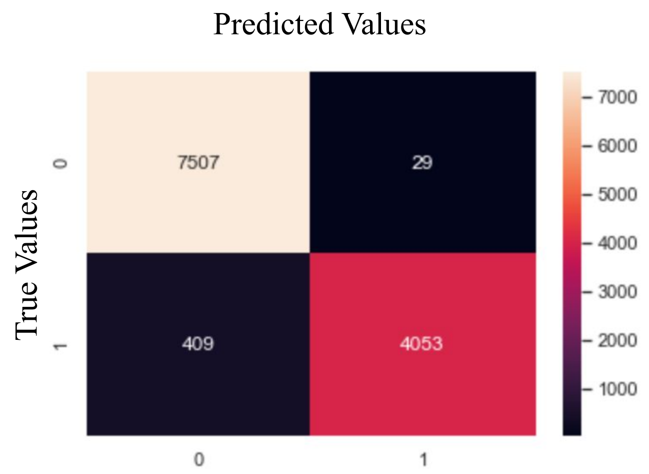
```
              precision    recall  f1-score   support

           0       0.63      0.82      0.72      5137
           1       0.54      0.30      0.38      3480

    accuracy                           0.61      8617
   macro avg       0.59      0.56      0.55      8617
weighted avg       0.59      0.61      0.58      8617
```



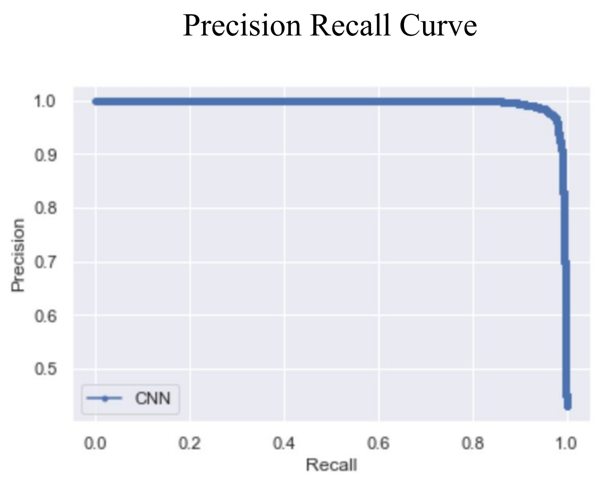
- The results of the CNN on the test set were quite negative. Using a threshold of 0.5, the accuracy was 0.61 and the weighted average f-1 score was 0.58.
- However, the recall for the fire class was 0.82, which was good as the model was not producing many false negatives because in the case of forest fires false negatives are very costly.

Convolutional Neural Network (After Custom Split)

- 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.
- There's a significant decrease in number of false positives from the initial testing results



```
              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
```



Discussion

- The autoencoder's inability to extract small fires 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, the autoencoder only reconstructed the images still keeping 32,000+ features.
- 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.
- 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.
- Wrongly assuming that my model's major problem was overfitting allowed me to experiment with trying to reduce overfitting which ended up being a blessing in disguise that taught me the importance of batch normalization and simpler models.

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