FINAL REPORT: FIRE DETECTION

EmberOptics

Jessica Johnson

Prerak Mehta

Fabien Rodriguez

Shabieh Saeed

Table of Contents

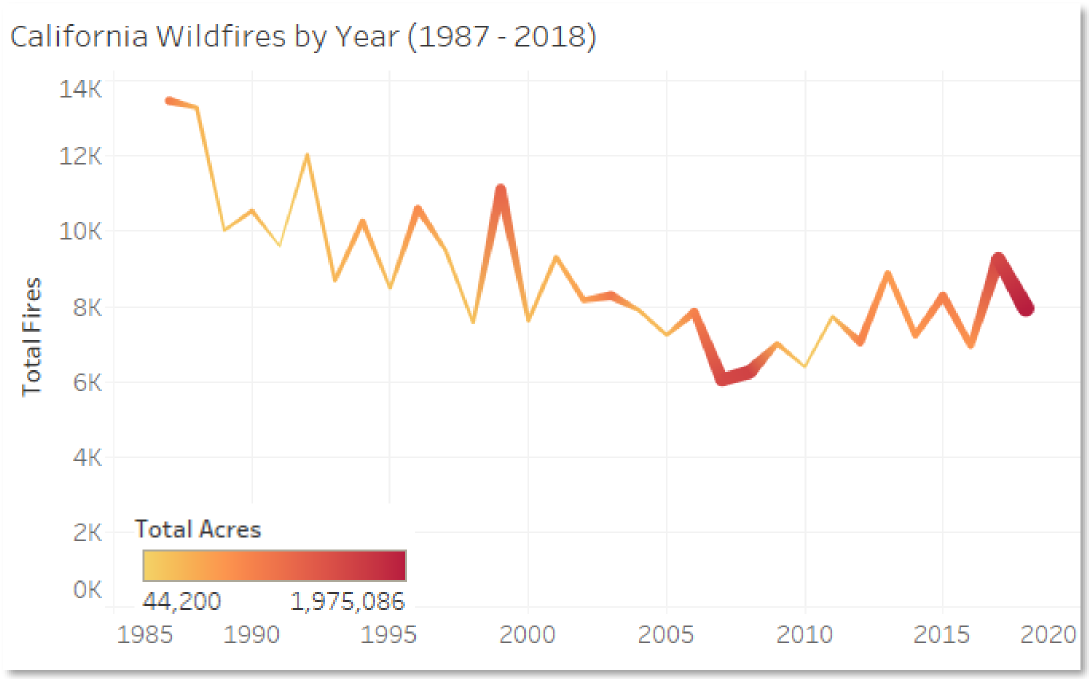
OVERVIEW

A wildfire is an unplanned fire that burns in a natural area such as a forest, grassland, or prairie. Wildfires are often caused by human activity or a natural phenomenon such as lightning, and they can happen at any time or anywhere. In 50% of wildfires recorded, it is not known how they started. The risk of wildfires increases in extremely dry conditions, such as drought, and during high winds. Wildfires can disrupt transportation, communications, power and gas services, and water supply. They also lead to a deterioration of the air quality, and loss of property, crops, resources, animals, and people.

The size and frequency of wildfires are growing due to climate change. Hotter and drier conditions are drying out ecosystems and increasing the risk of wildfires. Wildfires also simultaneously impact weather and the climate by releasing large quantities of carbon dioxide, carbon monoxide and fine particulate matter into the atmosphere. Resulting air pollution can cause a range of health issues, including respiratory and cardiovascular problems. Another significant health effect of wildfires is on mental health and psychosocial well-being.

Over the past couple of decades, the average number of annual wildfires in the United States has decreased; however, the average number of acres burned increased by an estimated 192,000 per year over the same timeframe.[1] This uptick in acreage burned is particularly pronounced in the western United States. Per the chart on the following page, the number of Wildfires in recent years in the state of California have started to go down though the acres of forest area they have been destroying have spiked up.

[1] https://www.rff.org/publications/explainers/wildfires-in-the-united-states-101-context-and-consequences/



Source: California Department of Forestry and Fire Protection; size and color of the line indicative of number of acres

Early detection of wild forest fires can reduce their severity and can help predict behavior. This is advantageous for regulating the fire as well as from a resource allocation perspective. Most importantly, enhanced wildfire detection can reduce the impact to human life. This project aims at building a machine learning model that predicts whether there is an active fire, no fire, or if a fire is about to start by processing an image of a forest.

BUSINESS CASE

EmberOptics' end goal in this project is to create a forest wildfire predictive model that can be programmed to process images taken by surveillance cameras, drones, and satellites. This project also aims at creating a dashboard that provides various analytics based on data retrieved by satellites. The dashboard will potentially show the geographical analysis of wildfires, along with its intensity, causes and seasonal analysis. Wildfires do occur naturally from lightning strikes or from the sun, however recent data shows that most are started by human carelessness. Campfires, cigarette buds, arson are responsible for 84% of wildfires started where the cause of the wildfire is known. Wildfires caused by human errors have tripled the fire season from 46 days to 154 days and cost up to $2 billion dollars every year.[2]

Given the sound business case, there are multiple options to be explored to monetize on this solution. For example, EmberOptics can generate revenue by selling the research and product to a third party, creating a business model and a clientele, or partnering up with the government. Due to the complexity associated with this project, the initial scope will focus only on the United States. Once the technology and analytical solutions are deployed and refined, we recommend scaling up the project so that coverage includes other countries prone to wildfires.

[2] https://untamedscience.com/blog/the-environmental-impact-of-forest-fires/

PROJECT OBJECTIVES

1. Leverage Computer Vision to keep an eye on forests by identifying areas of high risk and providing a proactive system to alert resources to changes in the ecosystem.

2. Use AI to detect fire and smoke by using image processing through an edge computing device in conjunction with a deploying a predictive model that assigns a confidence rate based on various attributes obtained from satellite images.

3. Create an interactive dashboard and mobile app for central monitoring, prediction and alerts for Forestry agencies, firefighters, and government offices.

4. Use AI to constantly update and train systems using the powerful predictive influence of artificial intelligence at our fingertips or in technological proximity where required.

PROJECT APPROACH

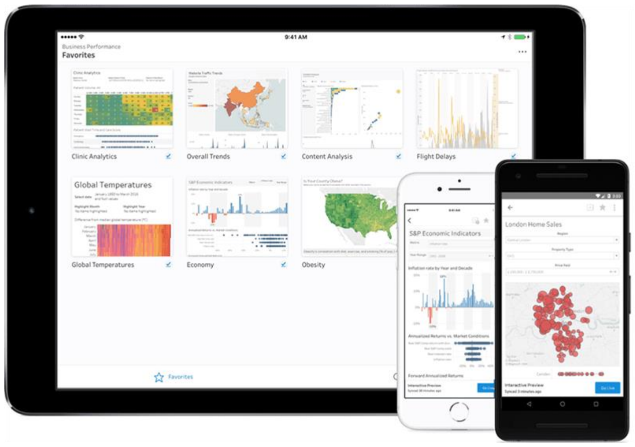
The EmberOptics team leveraged a dataset containing nearly 6,000 images of forest or forest-like environments divided into three categories: 'fire', 'non- fire', 'smoke'. Images labeled 'fire' contain visible flames, 'start fire' images contain smoke indicating the start of a fire. Finally, images labeled 'no fire' are images taken in forests. The data was split into a training set and test set. Convolutional neural network (CNN) and You Only Look Once (YOLO) models were built and trained to detect the presence or the start of a forest fire in an image. This model can ultimately be applied to detect a fire or a start of a fire from (aerial) surveillance footage of a forest. The model can be applied in real-time to low-framerate surveillance video (with fires not moving very fast, this assumption is somewhat sound) and give alert in case of fire. The training dataset was used to fit the models and the test dataset was used to estimate prediction error for model selection. The best model was selected and applied to the test dataset for final evaluation. The models were assessed using statistical evaluation criteria such as F1 scores and incorporated cross-validation to avoid overfitting.

The data has been developed through exploratory data analysis using Python. During this stage, we analyzed predictor relationships, cleaned the data, created new variables, transformed the data, addressed outliers (where needed), and explored clustering. The team experimented with various forms of modeling techniques using CNN and YOLO that are both interpretable and easy to explain.

The team also created a predictive model that assigned a confidence rate of a forest fire based on various attributes obtained from satellite images. This dataset comes directly from the NASA website. Since it is not possible to link our images with the data points collected in NASA's dataset, we have used two separate datasets to build our two models- 1) a dataset consisting forest and forest fires images to train our CNN and YOLO models, and 2) a tabular dataset consisting of forest fire related attributes collected by NASA satellite(s). Of course, our future goal beyond this project is to produce/collect a data bank that contains forest fire images and their respective attributes all in one place.

A user-friendly interactive dashboard and mobile interface has been developed using Tableau, which will allow users to assess the various risks and predicted results based on real-time data retrieved from satellites once the full system has been deployed. The dashboard is highly customizable so that it can be adjusted based on user requirements. Our default view includes high-level visual analytics of forest fires by predictive confidence value and also has embedded filtering and highlighting capabilities that can be used to streamline visuals based on geographic location and confidence level. For optimal deployment, the EmberOptics team recommends that customers couple our solution with Tableau Server so information can be quickly assimilated throughout the organization.

Our mobile interface is a critical component to our solution as it will allow organizations to access our visuals and receive alerts essentially anytime, anywhere. Please note that the image to the right is illustrative only, intended to depict Tableau interface renderings on various devices.



Furthermore, another app has been developed that incorporates our image recognition methodology that can be applied to livestream video feeds from various sources. For the time-being, however, this app requires users upload images and/or videos until we are able to fully deploy our solution and connect to satellites, surveillance videos, drones, etc.

FIRE CONFIDENCE LEVEL PREDICTION MODEL

Data Overview

The dataset we selected was retrieved directly from the NASA website. The Near Real Time (NRT) data is collected by two instruments named Moderate Resolution Imaging Spectroradiometer (MODIS) and Visible Infrared Imaging Radiometer Suite (VIIRS) which are both later processed by NASA's Land, Atmosphere Near real-time Capability for EOS (LANCE) for Fire Information for Resource Management System (FIRMS). The thermal anomalies/active fire data point represent the center of a 1 km pixel that is flagged by both instruments. The data gets processed by LANCE within 30 minutes to 2 hours after they are downloaded from satellites.

The below are the attributes of the dataset along with their description. We plan to build a machine learning model that predicts the confidence rate of the fire.

Attribute	Short Description	Long Description
Latitude	Latitude	Center of 1km fire pixel but not necessarily the actual location of the fire as one or more fires can be detected within the 1km pixel.
Longitude	Longitude	Center of 1km fire pixel but not necessarily the actual location of the fire as one or more fires can be detected within the 1km pixel.
Brightness (MODIS) Bright_T14 (VIIRS)	Brightness temperature 21 Kelvin Brightness temperature 4 Kelvin	Channel 21/22 brightness temperature of the fire pixel measured in Kelvin. Channel 4 brightness temperature of the fire pixel measured in Kelvin.
Scan	Along Scan pixel size	The algorithm produces 1km fire pixels but MODIS pixels get bigger toward the edge of scan. Scan and track reflect actual pixel size.

Predictive model dataset attributes continued…

Attribute	Short Description	Long Description
Track	Along Track pixel size	The algorithm produces 1km fire pixels but MODIS pixels get bigger toward the edge of scan. Scan and track reflect actual pixel size.
Acq_Date	Acquisition Date	Data of MODIS acquisition.
Acq_Time	Acquisition Time	Time of acquisition/overpass of the satellite (in UTC).
Satellite	Satellite	A = Aqua and T = Terra.
Confidence	Confidence (0-100%)	This value is based on a collection of intermediate algorithm quantities used in the detection process. It is intended to help users gauge the quality of individual hotspot/fire pixels. Confidence estimates range between 0 and 100% and are assigned one of the three fire classes (low-confidence fire, nominal-confidence fire, or high-confidence fire).
Bright_T31	Brightness temperature 31 (Kelvin)	Channel 31 brightness temperature of the fire pixel measured in Kelvin.
FRP	Fire Radiative Power (MW - megawatts)	Depicts the pixel-integrated fire radiative power in MW (megawatts).
Type*	Inferred hot spot type	0 = presumed vegetation fire 1 = active volcano 2 = other static land source 3 = offshore
DayNight	Day or Night	D= Daytime fire, N= Nighttime fire

The above dataset is available for 211 countries dating back to the year 2000. Due to the excessive amount of data available we decided to have our model focus only on the data acquired for the USA for now. The combined data retrieved for just the United States from both instruments MODIS and VIIRS itself is over 7.5 million rows. However, there are minor differences in the data collection process and data attributes. The biggest of which is prediction variable i.e. Confidence Level. MODIS dataset represents the confidence level of the fire with a value from 0-100%. This value is determined by NASA based on a collection of intermediate algorithm quantities used in the detection process. Where VIIRS dataset classifies confidence level into three categories - Low confidence fire, Nominal confidence fire, or High-confidence fire. Another difference in the two datasets is in brightness metric/channels used to detect the brightness coming from a potential forest fire image. And lastly but very importantly the difference in dataset size is significant. Although the MODIS dataset dates back to the year 2000, it is less than half the size of the VIIRS dataset that only dates back to the year 2012 (approx. 2.5M rows vs 5.2M rows). This was a considerable factor not only because of the difference in total size of the data but also the recency of the data.

It was imminent to find out which dataset contained more useful data and promised a better prediction accuracy as only one of them could be used for our final model. Also our future data collection process will need to reflect the similar process and attributes collected in the final dataset we built our model on. Based on just the eye test and insights from the comparison between the two datasets, the VIIRS dataset definitely looked more promising.

In order to test which dataset was better we created two models: 1) Random Forest Regression model for the MODIS dataset and 2) XGBoost Classification Model for the VIIRS dataset. The data pre-processing steps for both models were almost the same as the attributes were very much alike.

Model Approach

Pre-Processing:

1. Concatenate all the data files for each year into one panda dataframe for each of the two instruments.
2. Investigate data types of all columns and then for any null values. Investigate the number of recurring values in individual attributes to see which attributes are invaluable towards building a predictive model.
3. Build a heat map to view the correlation between attributes.
4. Convert alphabetical categorical attributes into numeric categorical data type columns using get_dummies and binning methods.
5. Break the timestamp column into year, month, and day columns.
6. Drop all columns concluded and analyzed as invaluable based on the steps above (In both case acq_date, acq_time, satellite, version, and instrument will be dropped)

Random Forest Regression Model:

- Split the cleaned MODIS dataframe into training and testing dataset and train a random forest regressor model on it. Evaluate the accuracy of the model (R2 score) on the test set using the in-built accuracy method of the random forest regressor function.

Random Forest Regression Model using Randomized Search CV:

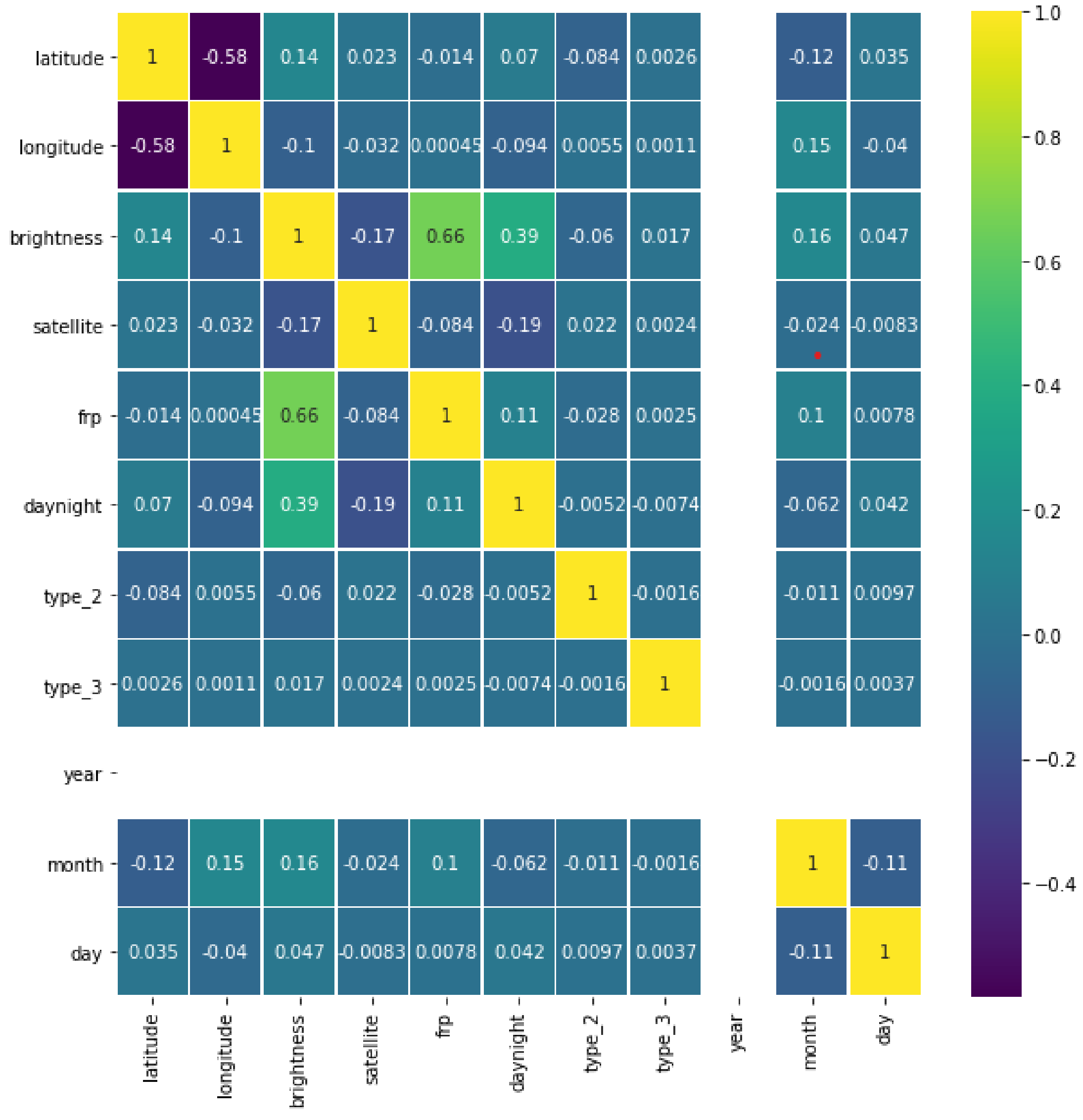
- Split the cleaned MODIS dataframe into training and testing dataset and train a Randomized Search CV model using a customized range of parameters (displayed in the code in the index) over 50 iterations.
- Once the best parameters are retrieved, train a random forest regressor model on those parameters and repeat the steps mentioned in the above section.

XGBoost Classification Model:

- Split the cleaned VIIRS dataframe into training and testing dataset and train an XGBClassifer model on it. Use a confusion matrix to spit out an accuracy score. Use K cross validation (10 folds) to become extra certain about the accuracy score.

Summary of Findings

The following is a heatmap to show the correlation between attributes of the dataset.



	latitude	longitude	brightness	satellite	frp	daynight	type_2	type_3	year	month	day
latitude	1	-0.58	0.14	0.023	-0.014	0.07	-0.084	0.0026		-0.12	0.035
longitude	-0.58	1	-0.1	-0.032	0.00045	-0.094	0.0055	0.0011		0.15	-0.04
brightness	0.14	-0.1	1	-0.17	0.66	0.39	-0.06	0.017		0.16	0.047
satellite	0.023	-0.032	-0.17	1	-0.084	-0.19	0.022	0.0024		-0.024	-0.0083
frp	-0.014	0.00045	0.66	-0.084	1	0.11	-0.028	0.0025		0.1	0.0078
daynight	0.07	-0.094	0.39	-0.19	0.11	1	-0.0052	-0.0074		-0.062	0.042
type_2	-0.084	0.0055	-0.06	0.022	-0.028	-0.0052	1	-0.0016		-0.011	0.0097
type_3	0.0026	0.0011	0.017	0.0024	0.0025	-0.0074	-0.0016	1		-0.0016	0.0037
year											
month	-0.12	0.15	0.16	-0.024	0.1	-0.062	-0.011	-0.0016		1	-0.11
day	0.035	-0.04	0.047	-0.0083	0.0078	0.042	0.0097	0.0037		-0.11	1

The random forest regressor model did not return a great accuracy and was also overfitting. Hence, we did some model tuning and used RandomCV to produce parameters for a random forest regressor model to give a little higher accuracy. However, both models produce almost identical numbers in terms of training and test accuracy scores i.e. training accuracy score ~ 95%/96% and testing accuracy score ~ 67%/70% respectively. This suggests considerable overfitting.

It is important to mention that an almost similar accuracy was derived in our initial findings where only 360,000 rows were used to train the model. On the contrary the amount of time taken to build the Randomized Search CV Model and get the best parameters took 5 times longer (almost 6 hours) on the full 2.5M (approximately) rows MODIS dataset

Because of such poor results we decided to explore another dataset (VIIRS) from the same source as the MODIS dataset and see if we could create a more robust model. We trained an XGBoost Classification model on the VIIRS dataset. It turned out that not only was the model producing a significantly higher test accuracy of over 98% but also took 1/5th of the time (compared to the RandomCV Model) to train on a dataset that was over twice as large. The model was also tested using K-Fold Cross Validation. The accuracy was almost the same and the standard deviation was negligible (0.02%). We had a clear winner with this model.

Here is the comparison of results received from the two methods (three models in total including Randomized Search CV):

	XGBClassifier	Random Forest Regressor	Random Forest Regression using Random Search CV parameters
Training Accuracy	98.35%	95.31%	96.1%
Testing Accuracy	98.3% (after K-fold)	67.39%	71.07%
Time Spent	1 hr 27 mins	1 hr 7 mins	5 hr 55 mins

CONVOLUTIONAL NEURAL NETWORK MODEL

Data Overview

The dataset used for the convolutional neural network (CNN) model contains approximately 6,000 unique images consisting of primarily woodland or forest-like pictures. The dataset included images with fires, images where no fire is present, and images with only smoke, potentially indicating the start of a fire. For testing and training purposes, we used the classification scheme of non-fire, fire, and smoke.

Model Approach

To achieve our image processing objective, the team first built a CNN model due to the algorithm's ability to detect important features without human intervention and/or supervision. We developed the network to train our images on using separable convolution. Furthermore, we utilized Keras Sequential API, a user-friendly API, to build our ConvNet with a RELU activation function as well as Batch Normalization and MaxPooling.

We created multiple stacks of CONV-RELU-POOL, thereafter we built the FCN head of our network, utilizing two FCN layers with Dropout as well as the softmax classifier. Notably, the team determined that recall—the number of images retrieved using the model—was more important than precision due to the nature of the business case. We believe it is far more critical to ensure we do fail to identify an actual fire vs. seeking to minimize the risk of a false positive. To date, however, our scores for both precision and recall have been quite strong.

Summary of Findings

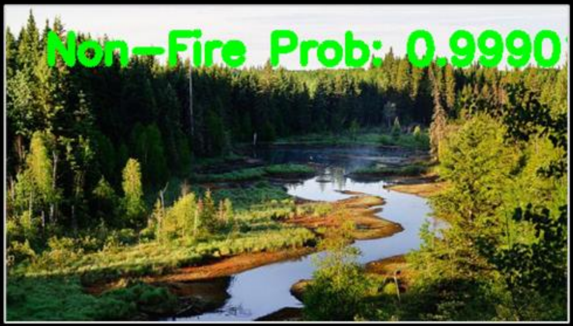
Our classifier was trained over 30 epochs, or cycles over the entire training dataset. We examined the multi-class F1 micro score to assess the performance of our classification model. The team was very pleased with our initial F1 micro score of 0.933 but have since explored alternate methods and further improved accuracy. The table below summarizes the performance of our initial model.

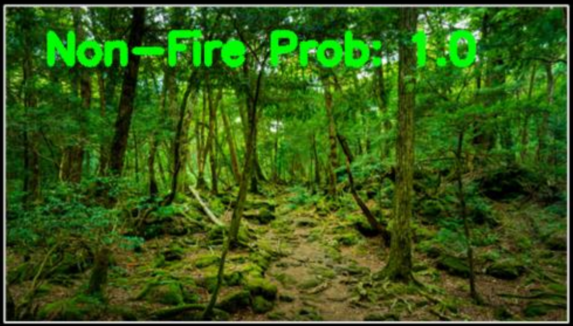
```
              precision   recall    f1-score    support

    Non-Fire       0.93      0.91      0.92        352
        Fire       0.88      0.98      0.93        303
       Smoke       0.95      0.94      0.94        509

    accuracy                           0.94       1164
   macro avg       0.92      0.94      0.93       1164
weighted avg       0.93      0.94      0.93       1164
```

While the team was very pleased with these early results, we recognized that there were opportunities for improvement and opted to execute additional models discussed in the next section. We are especially concerned with minimizing the number of false negatives, or instances where our model misses catching an ignition. There is also an opportunity to increase the probability associated with some of the images that are identified correctly. The images on the following page are examples of our model output that illustrate both correct image classification as well as incorrect classification.

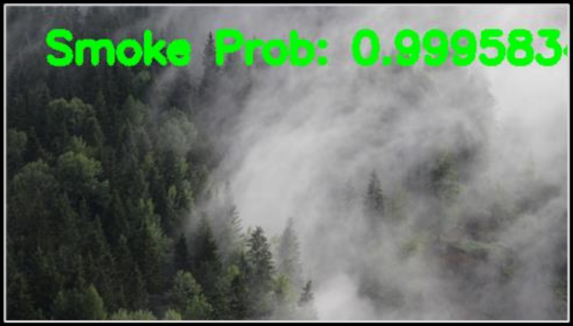
Correct Classification

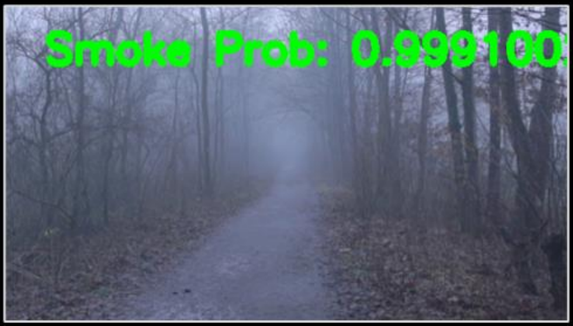
Non-Fire Images



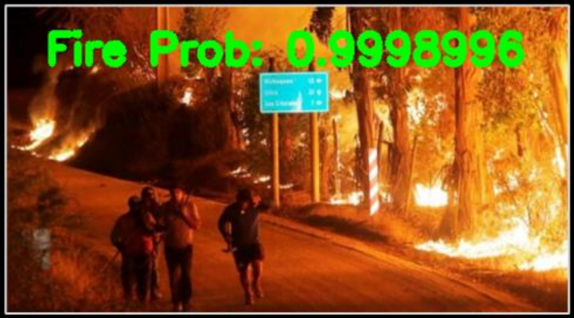


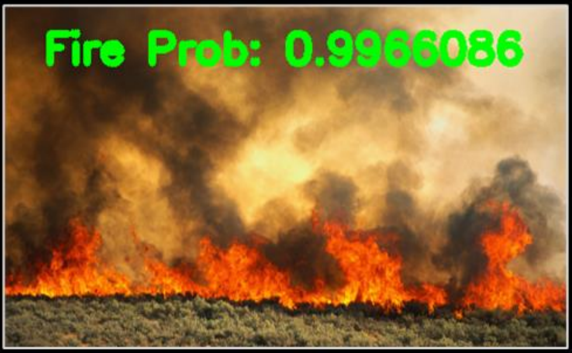
Smoke Images

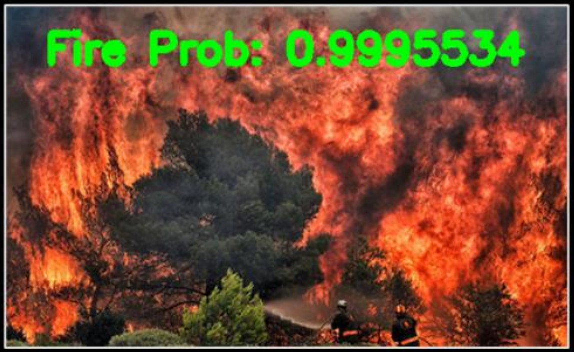


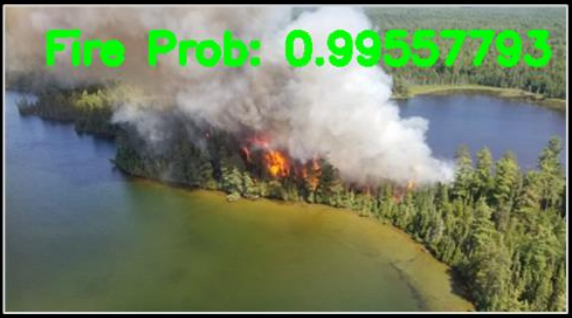


Fire Images

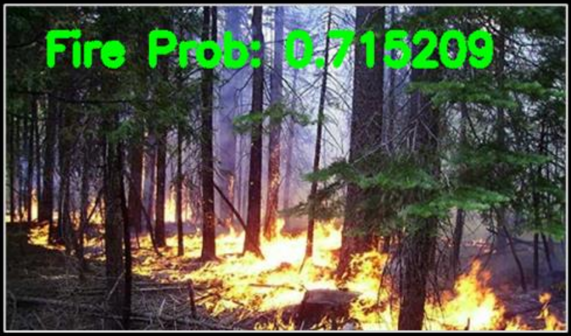


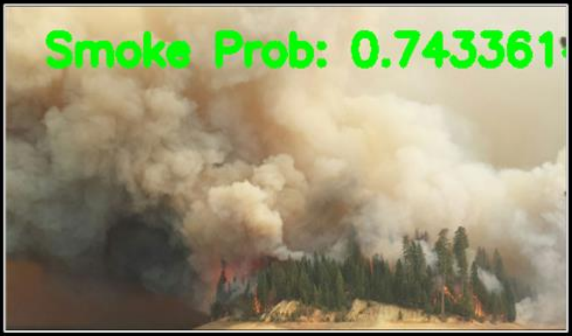






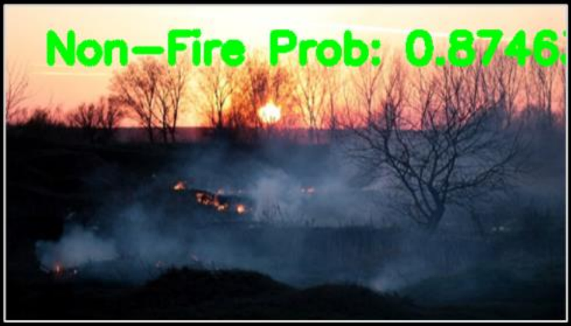
Opportunity Examples

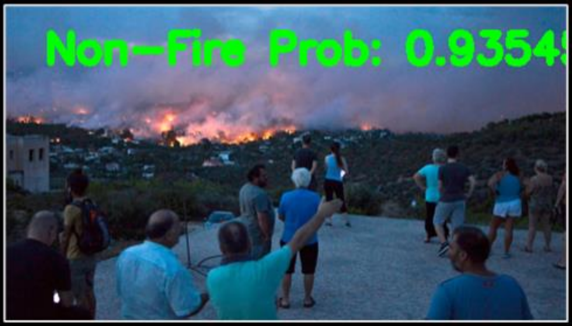




Both images are classified correctly, however the probability in these examples is below 0.75; there is a clear opportunity to improve the probability in these examples

Incorrect Classification





In both of the above instances fire is present yet the model assigned the non-fire classification with relatively high probability; we will seek to minimize this type of misclassification

YOU ONLY LOOK ONCE MODEL

Data Overview

The dataset used for the You Only Look Once (YOLO) model is the same dataset that was leveraged for our convolutional neural network. As a reminder, the dataset contains approximately 6,000 unique images consisting of woodland or forest-like pictures broken down into three classifications: images with fires, images where no fire is present, and images with only smoke, potentially indicating the start of a fire. We again used the classification scheme of non-fire, fire, and smoke.

Model Approach

After experimenting with CNN architectures we moved to regional CNN classification networks (fast RCNN) which improve upon standard CNN as they can detect the region where the object of interest occurs. YOLO is akin to a fully connected neural network (FCNN) and passes the image nxn once as this is a form of SSD (Single Shot Detector). We split the image by an mxm grid and for each bounding box we have a prediction for the presence of an object. Most Likely the bounding box is larger than the actual grids.

The object detection is reframed as a regression problem from pixels to bounding box coordinates and class probabilities.

A single CNN predicts for multiple bounding boxes and predicts for them, whereas YOLO trains on full images and the goal is to maximize performance. There are several benefits for using YOLO compared to traditional object detection systems. YOLO is much faster to train compared to CNN. As frame detection is framed as a regression problem we can simplify the pipeline. We run our FCNN during test time to gather our predictions. The base network runs between 25-150 FPS, so YOLO can be used for real time object detection.

YOLO also sees the global image during train and test time so it has a better understanding of contextual presence of objects, as opposed to sliding window/regional approaches. Fast R-CNN often mistakes backgrounds for objects as it does not have the benefit of knowing context. YOLO has almost 50% greater performance compared to fast RCNN and traditional CNN methods.

YOLO uses features from the entire image to predict the bounding boxes for each class. We can simultaneously predict the bounding box for each class. The design enables real time speed with high accuracy and precision.

The YOLO system works with the image being divided into an S x S grid. The grid cell is responsible for detecting any object whose center is in the center. The grid cell predicts a finite number of bounding boxes and confidence scores for those boxes. The confidence scores determine the certainty of the object being contained in the box. If no object exists within the cell then we don't predict. The confidence score is defined as P(obj)*IOU, where IOU is the Intersection Over Union. You can think of IOU as the degree of accuracy of a predicted bounding box and the ground truth bounding box. You can imagine that if the prediction is completely overlapping with the ground truth, then IOU is 100%, however if they completely deviate and there is no overlap, it will be 0%, and as the two deviate the degree of overlap will increase the metric.

The bounding boxes have five elements that are predicted: x, y, w, h, and confidence. The x and y represent the center of the bounding box, w and h are the width and height and confidence is the confidence of the class of the box. The confidence prediction is the IOU of ground truth box and prediction. Each grid cell predicts the conditional class probabilities. The probabilities are P(C|Obj) where C is the object. There can only be one class per box, which is the argmax.

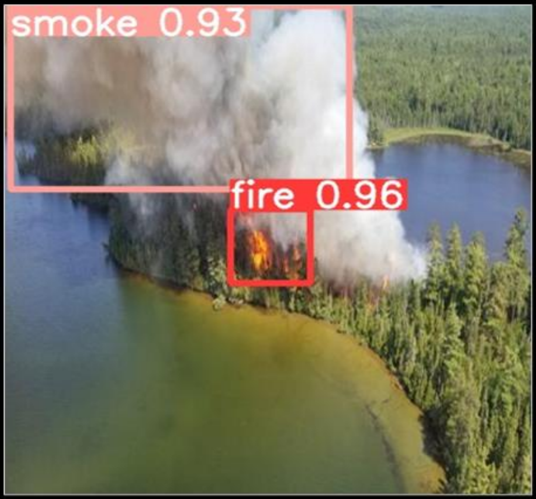
Summary of Findings

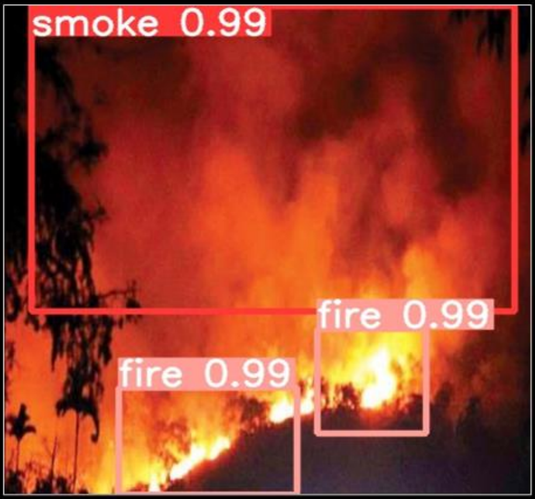
We used Roboflow to train the model using 2,000 pictures of fire, smoke and regular scenery. We had to annotate these photos to show the model where the fire/smoke lay and we trained it over 50 epochs to maximize mAP at IOU50 threshold. We saved the model trained on these photos and used it to predict fire/smoke in the results. We were able to see the accuracy mentioned above and precisely were able to match our predictions with the ground truth box. The below table contains the performance summary of our YOLO model.

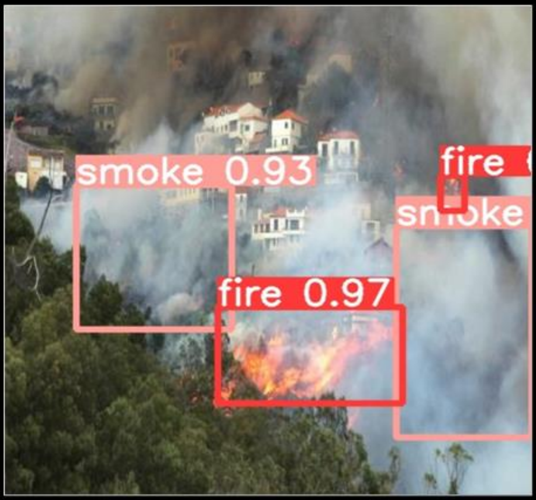
```
                precision  recall   f1-score   support

    Non-Fire       0.99      1         0.99       352
        Fire       0.99      0.98      0.98       303
       Smoke       0.99      0.97      0.98       509

    accuracy                           0.99      1164
   macro avg       0.99      0.97      0.98      1164
weighted avg       0.98      0.99      0.98      1164
```

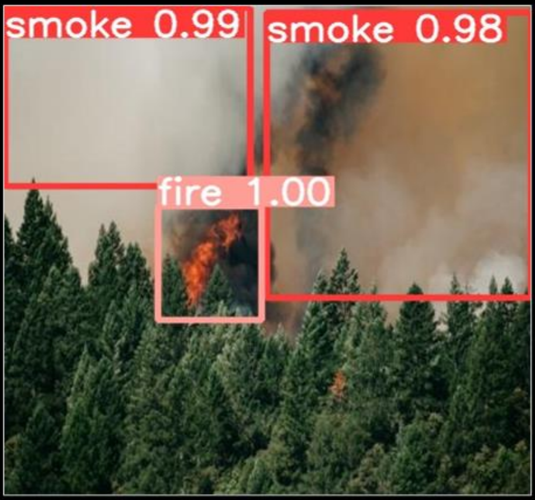
When compared to our convolutional neural network, the YOLO model is more accurate. The team feels very confident in the YOLO object detection method we developed and are recommending this model be incorporated into our final product. The next couple of pages contain examples of the YOLO model output.

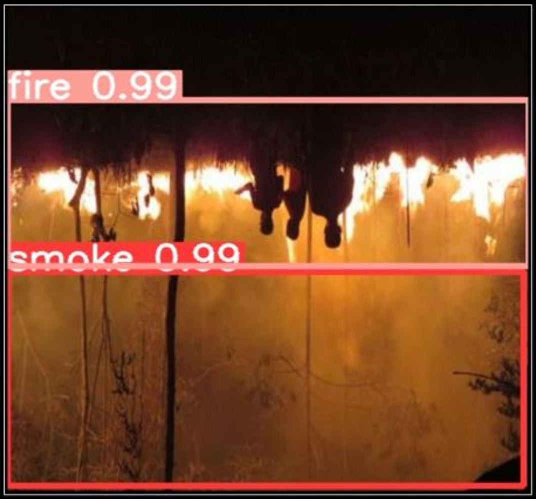
YOLO Examples





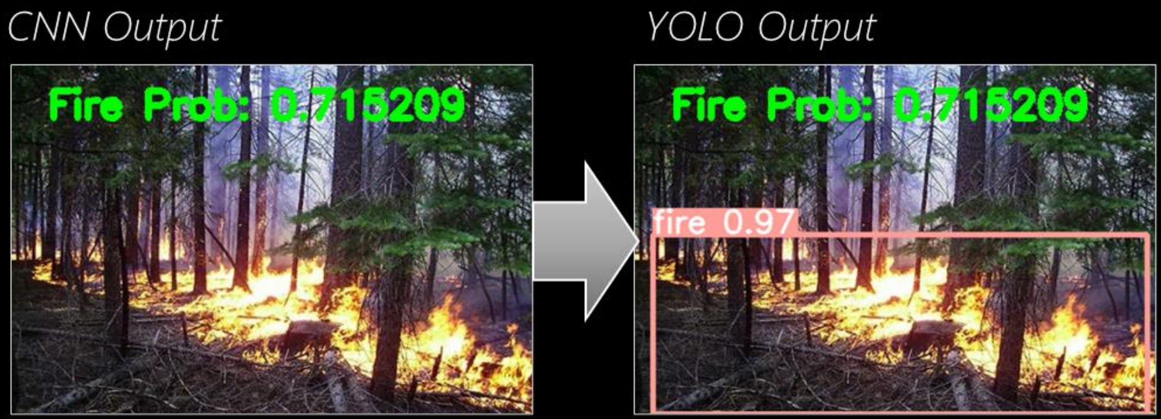




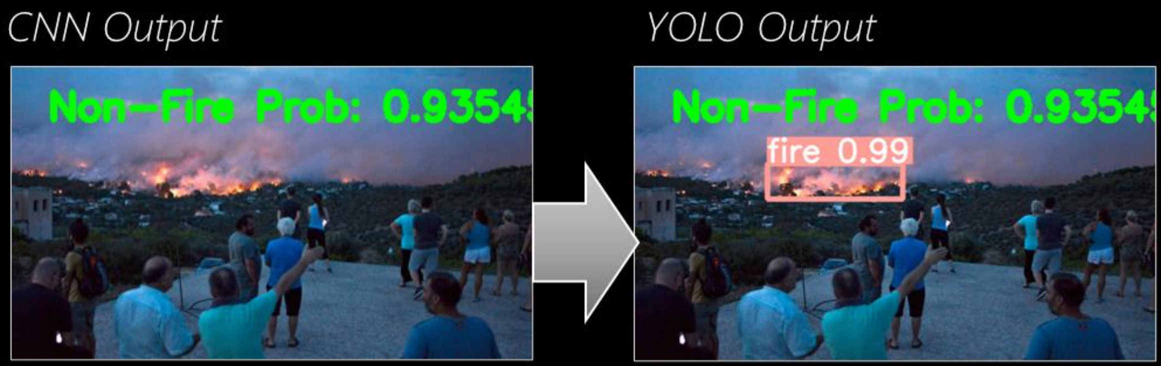


The team manipulated the orientation of the images to enhance accuracy and ensure model could operate effectively despite camera angle. The example to the left shows highly accurate results despite the picture rendering upside down.

Opportunities Identified with CNN



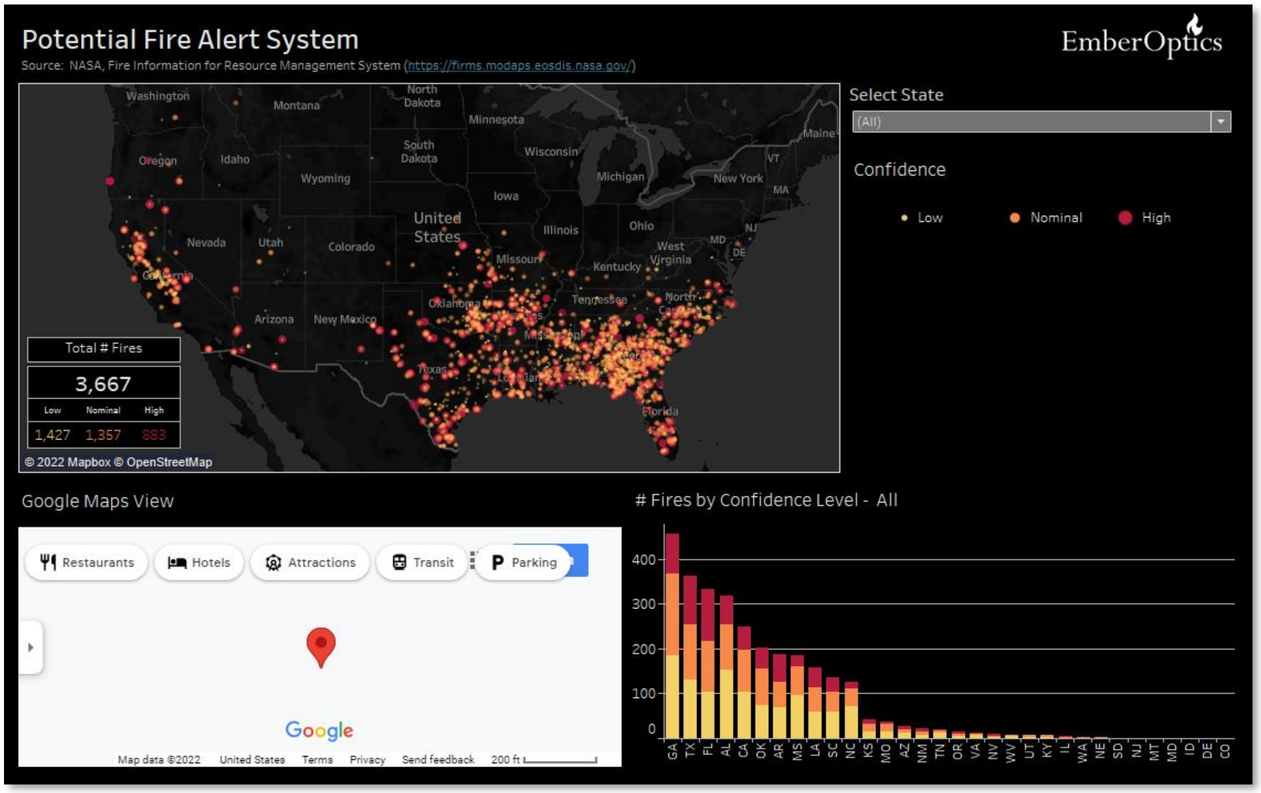
CNN Incorrect Classification



INTERACTIVE DASHBOARD

Impactful and intuitive data visualization for the end user will be an essential part of our final product. The EmberOptics team has created data visualizations that illustrate the results of our models. The below image captures the current version of our dashboard and, while near completion, will need to establish connection to live satellites in order to exercise all functionalities included. In particular, the aerial image found on the subsequent pages is an illustrative view of what a user can expect to see once EmberOptics is able to retrieve real time images from satellites. The remaining functionality in the current dashboard is fully operational and is presently leveraging seven (7) days of historical thermal detections provided by the NASA Fire Information for Resource Management System (FIRMS).

Below is the default view of the dashboard. All potential fires will be displayed with mark color and map point size indicative of the level of confidence associated with our regression model. All incidents are geospatially depicted in the map view with a summary of all potential fires in the contiguous United States as well the number of fires by state.



Potential Fire Alert System

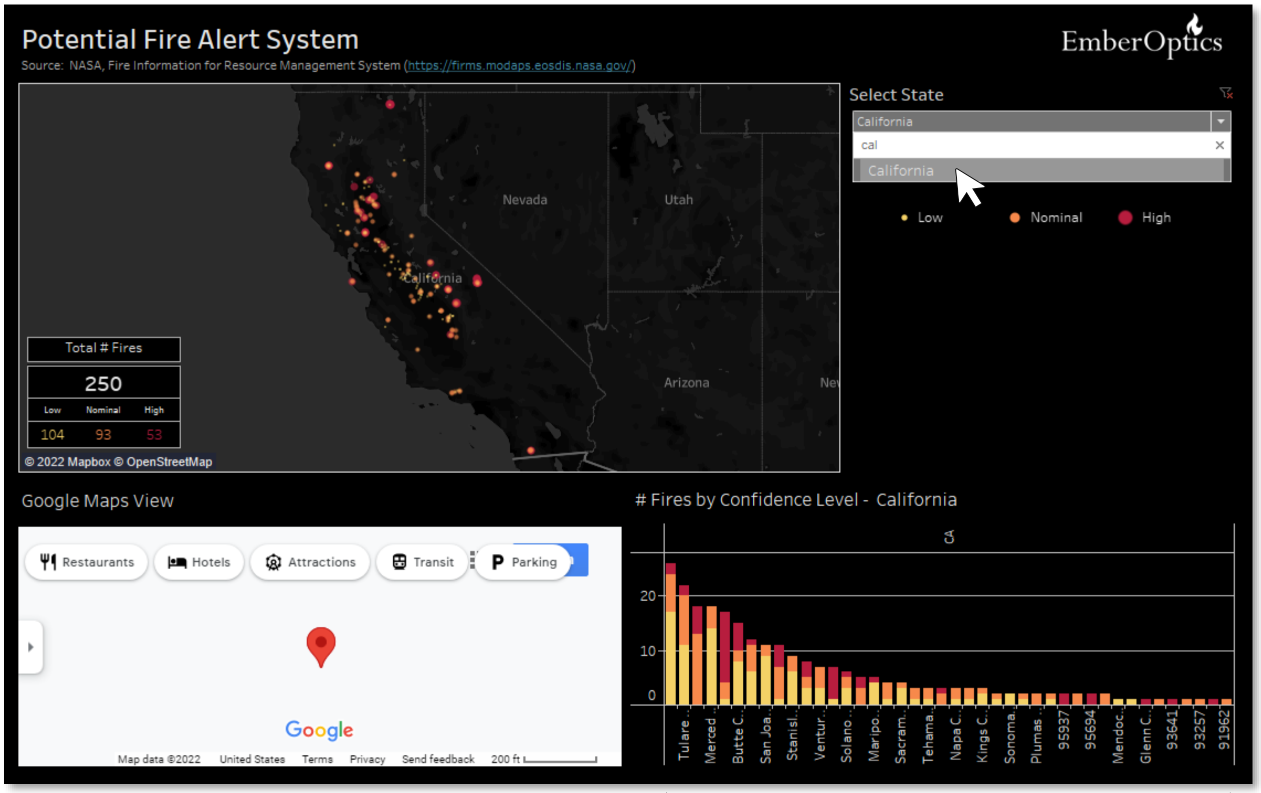
Source: NASA, Fire Information for Resource Management System (https://firms.modaps.eosdis.nasa.gov/)

Select State: (All)

Total # Fires		
3,667		
Low	Nominal	High
1,427	1,357	883

Users are easily able to filter on specific states to narrow the map view. This filter is connected to the small table summarizing the total number of potential fires. Additionally, the filter also updates the stacked bar chart so that it automatically shows the number of fires by county for the state selected, sorted from highest number to lowest. The current iteration of the dashboard has a filter for states only but can be easily modified to include other geographic attributes such as city or county depending on end user preference. For this dashboard we opted to keep the view simple in lieu of incorporating multiple interactive fields that may prove confusing to users.

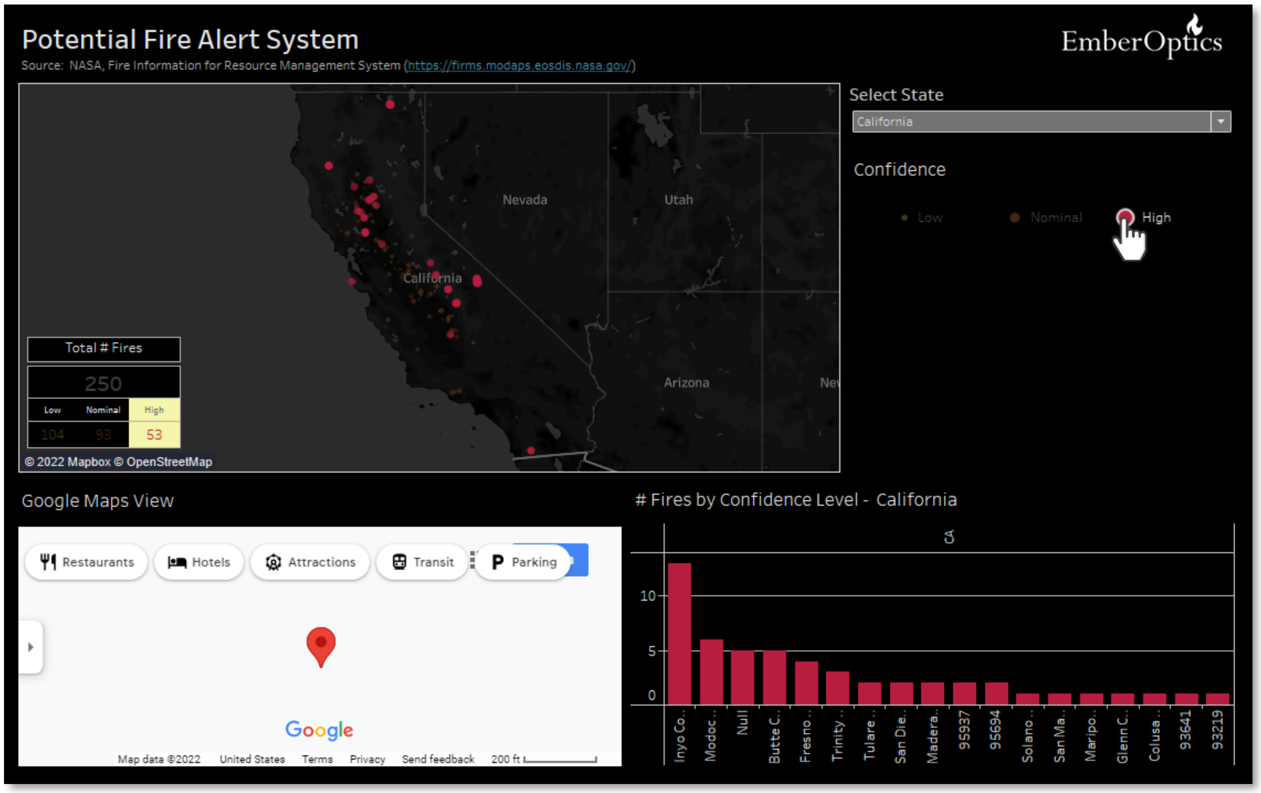
State Filter Example:



Selecting a specific state will update this view to # of fires by county

Another interactive feature incorporated into the dashboard is the highlight functionality where users can click on a specific confidence classification group, focusing on only the desired subset of potential fires based on user selection.

Highlight Based on Confidence Level Example:

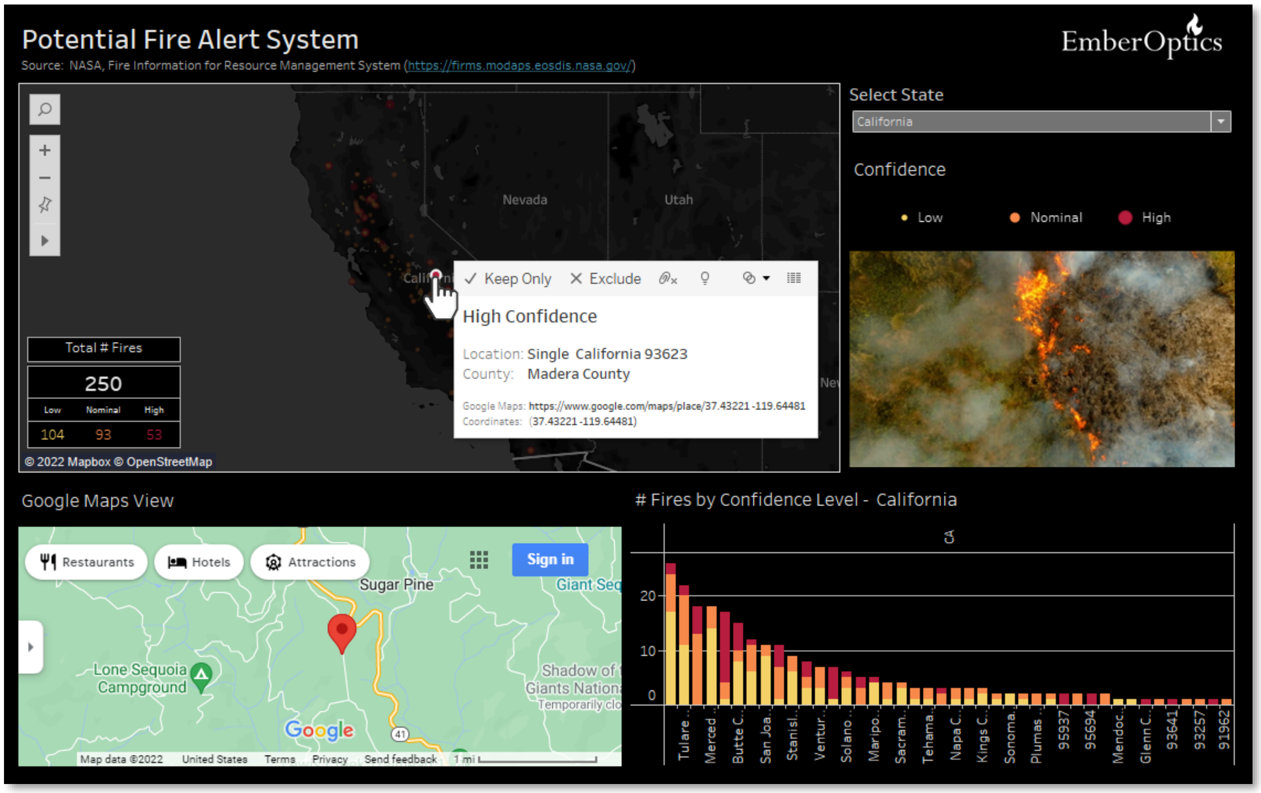


In this dashboard, a classification scheme of "Low Confidence", "Nominal Confidence", and "High Confidence" was used; however, these naming conventions, colors, and sizes can be easily customized per end user requirements. As evidenced in the above image, highlighting a specific confidence level will filter the bar chart and sort counties by number of potential fires within the confidence grouping. All information will render again upon deselecting confidence level.

Arguably the most important feature embedded in this dashboard is the ability to assess a specific incident. Users can click on a single mark on the map; a tooltip will appear and renders the assigned confidence classification (in this example "High Confidence") as well as available locality information including street address and county as well as the GPS coordinates. Google Maps has been embedded into the dashboard and pinpoints the exact location of the potential fire for each mark on the map, making it easy for the end user to get directions to the incident.

As alluded to previously, the image shown on the right-hand side is illustrative only at this time. Once EmberOptics is connected to live satellites these images will represent a real-time aerial view of the location, enabling the user to get visual confirmation of the fire and deploy resources accordingly.

Specific Location Data and Visuals Example:



Potential Fire Alert System

Source: NASA, Fire Information for Resource Management System (https://firms.modaps.eosdis.nasa.gov/)

Select State: California

High Confidence

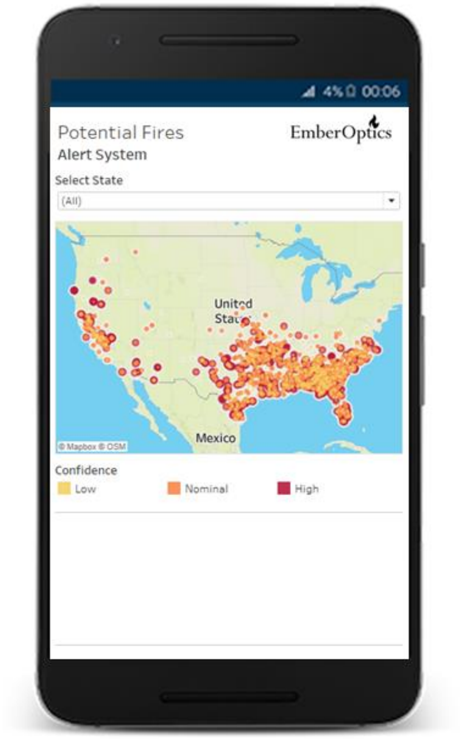
Location: Single California 93623
County: Madera County

Google Maps: https://www.google.com/maps/place/37.43221 -119.64481
Coordinates: (37.43221 -119.64481)

Total # Fires		
250		
Low	Nominal	High
104	93	53

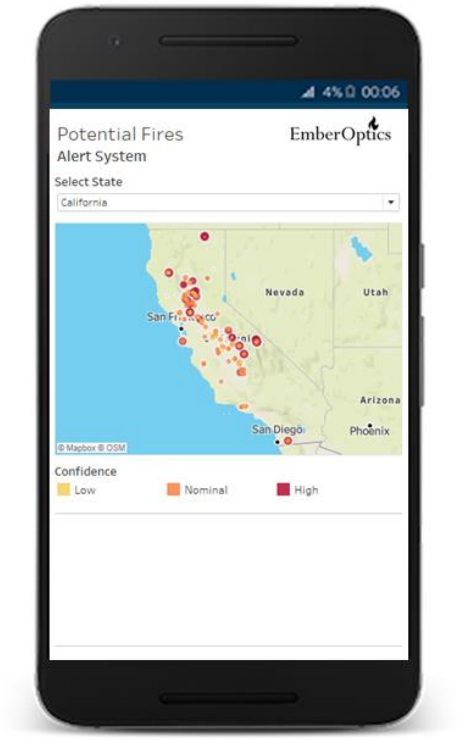
MOBILE APPS

Mobile Dashboard



Similar to the draft dashboard, the mobile phone dashboard contains an interactive map depicting potential fire ignitions. App users have the ability to navigate the entire United States, zooming in on specific geographies or to view the exact location of a potential fire. More map details are visible when a user has zoomed into a smaller area.

Whereas the initial version of the app had limited functionality, the updated version includes underlying geographic data which enables the use of dynamic filters and more descriptive information on potential fires.

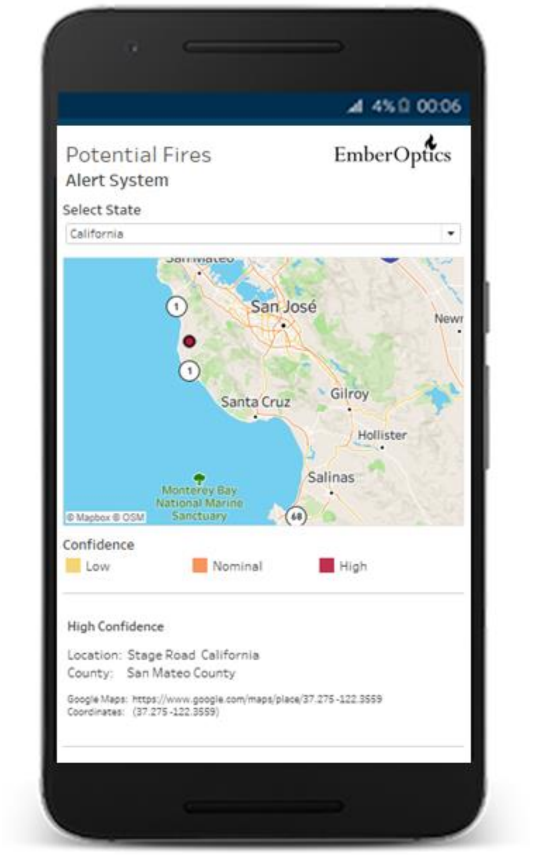


The above image is the default view upon opening the application. Currently there is only one filter built into the map which allows users to filter on state. This functionality is illustrated on the left where the state of California has been selected.

The same color scheme and confidence classification system has been applied to the mobile app for consistency purposes, so users are able to quickly view and consume information as they go from one platform to the other (from the desktop dashboard to the mobile app or vice versa).

In this final image of the mobile dashboard, map details such as location names, streets, and topography are more visible due to the level of zoom. When a user taps on a specific mark, or "bubble", location details will appear at the bottom of the screen including a specific Google Maps web address that can be followed to get directions to the incident.

As per our report recommendations, a later iteration of this app should include aerial images of the location when selected. This feature is currently available in the EmberOptics dashboard and could be highly useful as part of the mobile phone application as well.



Potential Fires
Alert System

Select State: California

Confidence
🟨 Low 🟧 Nominal 🟥 High

High Confidence

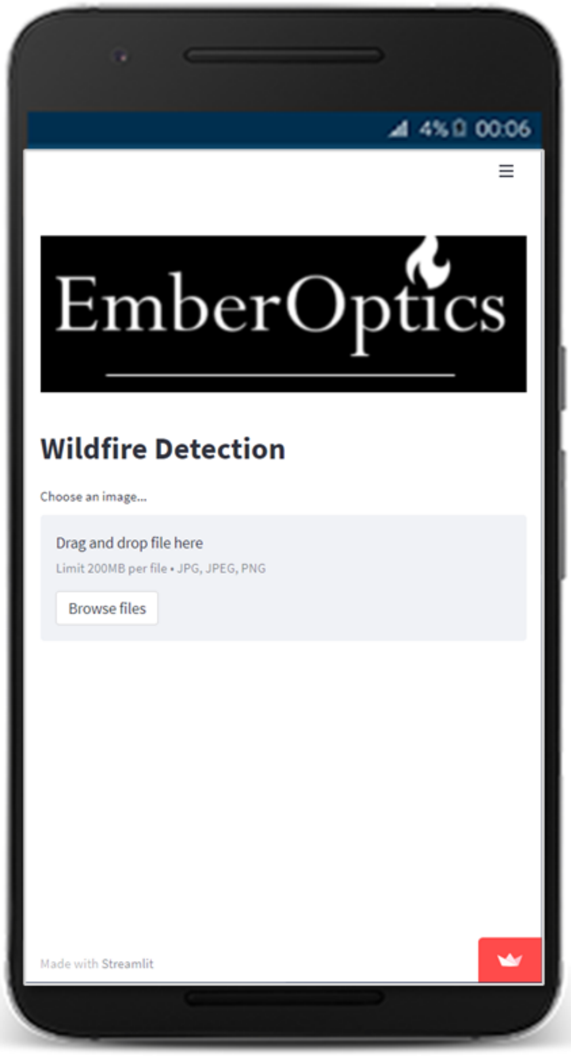
Location: Stage Road California
County: San Mateo County

Google Maps: https://www.google.com/maps/place/37.275-122.3559
Coordinates: (37.275 -122.3559)

Livestream App

In addition to the mobile dashboard the team has developed and deployed an application where users can upload images or videos that uses our YOLO model to identify the presence of fire and smoke. Although users currently are required to select a file upon accessing the app, once our solution is fully deployed we will connect this application to livestream videos from cameras, drones, and satellites. The below are screenshots of our application that depict what the user sees upon accessing the program and the model results, which render immediately, of the selected images and/or videos.

Landing Screen:



EmberOptics

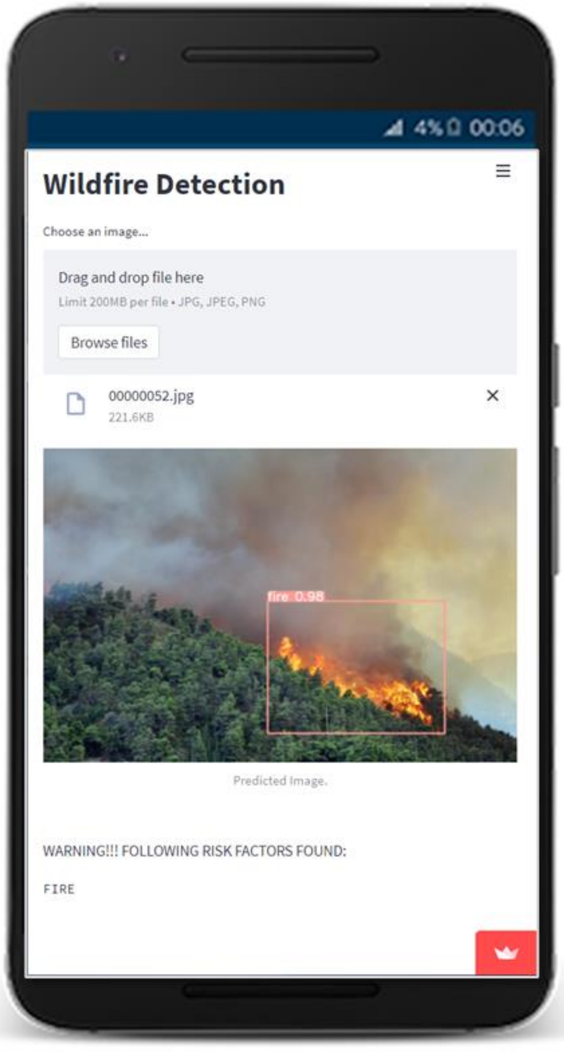
Wildfire Detection

Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Results Rendered on App:



Wildfire Detection

Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

00000052.jpg
221.6KB

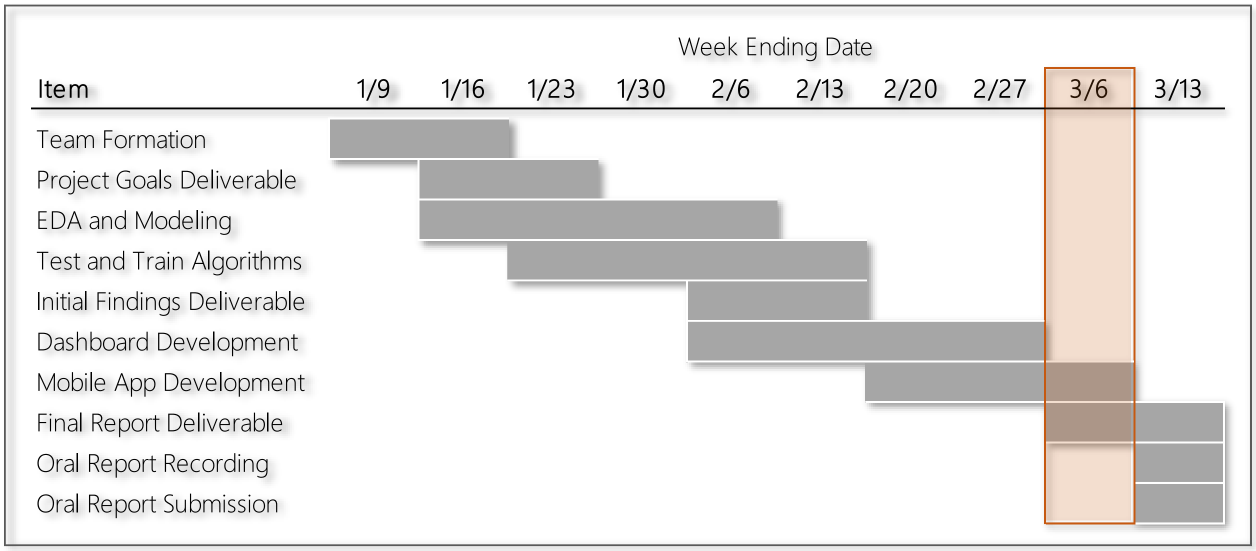
Predicted Image.

WARNING!!! FOLLOWING RISK FACTORS FOUND:

```
FIRE
```

PROJECT STATUS & MILESTONES

The EmberOptics project continues to progress according to schedule with all deliverables finalized and submitted on or before the due date. This Final Report and associated Executive Summary represents the conclusion of our analysis and solution development. At this juncture, the only outstanding item left to prepare is the final slide deck and oral presentation which is due in one week's time. Our team will now pivot to focusing 100% of our efforts on these last deliverables.



Item	Description
Team Formation	Engage with team, align on expectations, and select project
Project Goals Deliverable	Develop business case, objectives, project approach, and timelines
EDA and Modeling	Assess data and methods; finalize modeling approach and refine scope, if needed
Test and Train Algorithms	Develop models, assess accuracy and analyze results
Initial Findings Deliverable	Synthesize analysis and insights into report and executive summary
Dashboard Development	Create an interactive, client facing Tableau dashboard
Mobile App Development	Define requirements and outline app interface; create mock up
Final Report Deliverable	Compile comprehensive report of project and results; deliver executive summary
Oral Report Recording	Record team presentation and submit on Canvas
Oral Report Submission	Submit oral report, presentation, and executive summary slides

FINAL RECOMMENDATIONS

The EmberOptics team completed the refinement of the models through extensive testing of these interactions on the datasets. This testing resulted in the selection of the best model/s for integration into the final product. This final product includes an extensive interactive dashboard and mobile application.

The final product presents and documents all recommendations and is being used as the basis for the final presentation to the CEO. It includes details on the use of the CNN models and YOLO approach with bounding boxes.

The EmberOptics team sees this product as one of great potential, opportunity and benefit serving a much-needed role in terms of preservation of a healthy ecosystem that may endanger wildlife species, compromise air quality, and threaten the safety of all at risk communities.

FUTURE OPPORTUNITIES

- Streamline a reliable data collection process where potential forest fires images received from satellites/drones have the same attributes we used to train our classification model.
- Leverage XGBoost Classifier model together with YOLO to deploy highly accurate, predictive solutions.
- With more and more data available to us in the future, it is important to explore Big Data techniques and platforms that can crunch massive numbers and train machine learning models on humongous datasets.
- Continue to refine dashboard and mobile app to optimize visual impact; consider adding toggle functionality to both tools to preserve white space while giving the user the ability to enlarge certain graphics
- Broaden scope of solution beyond contiguous United States

EmberOptics

APPENDICES

TEAM QUALIFICATIONS

Jessica Johnson has nearly 15 years of experience working in diverse operations and business analytics roles and is currently the head of Business Intelligence at Altria Group. Jessica brings skills including strategy, management and business analysis.

Prerak Mehta is a Senior Data Analyst at Bloomberg law who has an experience of over 4 years in data analysis and engineering. Prerak brings in his skill set in exploring data and building machine learning models to the team.

Fabien Rodriguez is an IT professional who has held various roles in the field for over 20 years. Fabien brings a diverse skillset to the team with experience in leadership, planning and project management.

Shabieh Saeed is a Data Science Professional who has worked in the technology industry for over 10 years. Most recently his career has focused on NLP.

RANDOM FOREST REGRESSION MODEL CODE

```
Xtrain, Xtest, ytrain, ytest = train_test_split(fin.iloc[:, :500], y,
test_size=0.2)

random_model = RandomForestRegressor(n_estimators=300, random_state = 42,
n_jobs = -1)

random_model.fit(Xtrain, ytrain)
y_pred = random_model.predict(Xtest)
```

RANDOM FOREST USING RANDOMIZED SEARCH CV MODEL CODE

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 300, stop = 500, num =
20)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(15, 35, num = 7)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 3, 5]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
                }
print(random_grid)

rf_random = RandomizedSearchCV(estimator = random_model, param_distributions
= random_grid, n_iter = 50, cv = 3, verbose=2, random_state=42)
# Fit the random search model
rf_random.fit(Xtrain, ytrain)

rf_random.best_params_

random_new = RandomForestRegressor(n_estimators = 394, min_samples_split = 2,
min_samples_leaf = 1, max_features = 'sqrt',max_depth = 25, bootstrap = True)

random_new.fit(Xtrain, ytrain)
y_pred1 = random_new.predict(Xtest)
```

XGBCLASSIFIER MODEL CODE (INCLUDING K-FOLD CROSS VALIDATION)

```
%%time

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(fin, y, test_size = 0.2,
random_state = 0)

from xgboost import XGBClassifier

classifier = XGBClassifier()
classifier.fit(X_train, y_train)

%%time

from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)

%%time

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv
= 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

CONVOLUTIONAL NEURAL NETWORK CODE

```
class FireCNN:
  @staticmethod
  def build(width, height, depth, classes):
      # initialize the model along with the input shape to be
      # "channels last" and the channels dimension itself
      model = Sequential()
      inputShape = (height, width, depth)
      chanDim = -1

      # CONV => RELU => POOL
      model.add(SeparableConv2D(16, (7, 7), padding="same",
      input_shape=inputShape))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2, 2)))

      # CONV => RELU => POOL
      model.add(SeparableConv2D(32, (3, 3), padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2, 2)))

      # (CONV => RELU) * 2 => POOL
      model.add(SeparableConv2D(64, (3, 3), padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(SeparableConv2D(64, (3, 3), padding="same"))
      model.add(Activation("relu"))
      model.add(BatchNormalization(axis=chanDim))
      model.add(MaxPooling2D(pool_size=(2, 2)))

      # first set of FC => RELU layers
      model.add(Flatten())
      model.add(Dense(128))
      model.add(Activation("relu"))
      model.add(BatchNormalization())
      model.add(Dropout(0.5))

      # second set of FC => RELU layers
      model.add(Dense(128))
      model.add(Activation("relu"))
      model.add(BatchNormalization())
      model.add(Dropout(0.5))

      # softmax classifier
      model.add(Dense(classes))
      model.add(Activation("softmax"))

      # return the constructed network architecture
      return model
```

YOU ONLY LOOK ONCE CODE

```
import argparse
import logging
import math
import os
import random
import time
from copy import deepcopy
from pathlib import Path
from threading import Thread

import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

import test  # import test.py to get mAP after each epoch
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path,
labels_to_image_weights, init_seeds, \
      fitness, strip_optimizer, get_latest_run, check_dataset, check_file,
check_git_status, check_img_size, \
      check_requirements, print_mutation, set_logging, one_cycle, colorstr
from utils.google_utils import attempt_download
from utils.loss import ComputeLoss
from utils.plots import plot_images, plot_labels, plot_results, plot_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_dicts,
torch_distributed_zero_first, is_parallel
from utils.wandb_logging.wandb_utils import WandbLogger, check_wandb_resume

logger = logging.getLogger(__name__)

def train(hyp, opt, device, tb_writer=None):
      logger.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v
in hyp.items()))
      save_dir, epochs, batch_size, total_batch_size, weights, rank = \
      Path(opt.save_dir), opt.epochs, opt.batch_size, opt.total_batch_size,
opt.weights, opt.global_rank

      # Directories
      wdir = save_dir / 'weights'
      wdir.mkdir(parents=True, exist_ok=True)  # make dir
      last = wdir / 'last.pt'
```

```
        best = wdir / 'best.pt'
        results_file = save_dir / 'results.txt'

        # Save run settings
        with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.safe_dump(hyp, f, sort_keys=False)
        with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.safe_dump(vars(opt), f, sort_keys=False)

        # Configure
        plots = not opt.evolve  # create plots
        cuda = device.type != 'cpu'
        init_seeds(2 + rank)
        with open(opt.data) as f:
        data_dict = yaml.safe_load(f)  # data dict
        is_coco = opt.data.endswith('coco.yaml')

        # Logging- Doing this before checking the dataset. Might update data_dict
        loggers = {'wandb': None}  # loggers dict
        if rank in [-1, 0]:
        opt.hyp = hyp  # add hyperparameters
        run_id = torch.load(weights).get('wandb_id') if weights.endswith('.pt') and
os.path.isfile(weights) else None
        wandb_logger = WandbLogger(opt, save_dir.stem, run_id, data_dict)
        loggers['wandb'] = wandb_logger.wandb
        data_dict = wandb_logger.data_dict
        if wandb_logger.wandb:
                weights, epochs, hyp = opt.weights, opt.epochs, opt.hyp  #
WandbLogger might update weights, epochs if resuming

        nc = 1 if opt.single_cls else int(data_dict['nc'])  # number of classes
        names = ['item'] if opt.single_cls and len(data_dict['names']) != 1 else
data_dict['names']  # class names
        assert len(names) == nc, '%g names found for nc=%g dataset in %s' %
(len(names), nc, opt.data)  # check

        # Model
        pretrained = weights.endswith('.pt')
        if pretrained:
        with torch_distributed_zero_first(rank):
                attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location=device)  # load checkpoint
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc,
anchors=hyp.get('anchors')).to(device)  # create
        exclude = ['anchor'] if (opt.cfg or hyp.get('anchors')) and not opt.resume
else []  # exclude keys
        state_dict = ckpt['model'].float().state_dict()  # to FP32
        state_dict = intersect_dicts(state_dict, model.state_dict(),
exclude=exclude)  # intersect
        model.load_state_dict(state_dict, strict=False)  # load

        logger.info('Transferred %g/%g items from %s' % (len(state_dict),
len(model.state_dict()), weights))  # report
        else:
        model = Model(opt.cfg, ch=3, nc=nc,
anchors=hyp.get('anchors')).to(device)  # create
        with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
        train_path = data_dict['train']
        test_path = data_dict['val']

        # Freeze
        freeze = []  # parameter names to freeze (full or partial)
        for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
                print('freezing %s' % k)
                v.requires_grad = False

        # Optimizer
        nbs = 64  # nominal batch size
        accumulate = max(round(nbs / total_batch_size), 1)  # accumulate loss
before optimizing
        hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # scale
weight_decay
        logger.info(f"Scaled weight_decay = {hyp['weight_decay']}")

        pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
        for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
                pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
                 pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):
                pg1.append(v.weight)  # apply decay

        if opt.adam:
        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'],
0.999))  # adjust beta1 to momentum
        else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'],
nesterov=True)

        optimizer.add_param_group({'params': pg1, 'weight_decay':
hyp['weight_decay']})  # add pg1 with weight_decay
        optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
        logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' %
(len(pg2), len(pg1), len(pg0)))
        del pg0, pg1, pg2

        # Scheduler https://arxiv.org/pdf/1812.01187.pdf
        #
https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#OneCycleLR
        if opt.linear_lr:
        lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf']  #
linear


        else:
        lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']
        scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)
        # plot_lr_scheduler(optimizer, scheduler, epochs)

        # EMA
        ema = ModelEMA(model) if rank in [-1, 0] else None

        # Resume
        start_epoch, best_fitness = 0, 0.0
        if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
              optimizer.load_state_dict(ckpt['optimizer'])
              best_fitness = ckpt['best_fitness']

        # EMA
        if ema and ckpt.get('ema'):
              ema.ema.load_state_dict(ckpt['ema'].float().state_dict())
              ema.updates = ckpt['updates']

        # Results
        if ckpt.get('training_results') is not None:
              results_file.write_text(ckpt['training_results'])  # write
results.txt

        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if opt.resume:
              assert start_epoch > 0, '%s training to %g epochs is finished,
nothing to resume.' % (weights, epochs)
        if epochs < start_epoch:
              logger.info('%s has been trained for %g epochs. Fine-tuning for %g
additional epochs.' %
                          (weights, ckpt['epoch'], epochs))
              epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, state_dict

        # Image sizes
        gs = max(int(model.stride.max()), 32)  # grid size (max stride)
        nl = model.model[-1].nl  # number of detection layers (used for scaling
hyp['obj'])
        imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_size]  # verify
imgsz are gs-multiples

        # DP mode
        if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)

        # SyncBatchNorm
        if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        logger.info('Using SyncBatchNorm()')



        # Trainloader
        dataloader, dataset = create_dataloader(train_path, imgsz, batch_size, gs,
opt,
                                            hyp=hyp, augment=True,
cache=opt.cache_images, rect=opt.rect, rank=rank,
                                            world_size=opt.world_size,
workers=opt.workers,
                                         image_weights=opt.image_weights,
quad=opt.quad, prefix=colorstr('train: '))
        mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max label class
        nb = len(dataloader)  # number of batches
        assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels
are 0-%g' % (mlc, nc, opt.data, nc - 1)

        # Process 0
        if rank in [-1, 0]:
        testloader = create_dataloader(test_path, imgsz_test, batch_size * 2, gs,
opt,  # testloader
                                       hyp=hyp, cache=opt.cache_images and not
opt.notest, rect=True, rank=-1,
                                        world_size=opt.world_size,
workers=opt.workers,
                                       pad=0.5, prefix=colorstr('val: '))[0]

        if not opt.resume:
                labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes
                # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
                # model._initialize_biases(cf.to(device))
                if plots:
                plot_labels(labels, names, save_dir, loggers)
                if tb_writer:
                        tb_writer.add_histogram('classes', c, 0)

                # Anchors
                if not opt.noautoanchor:
                check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)
                model.half().float()  # pre-reduce anchor precision

        # DDP mode
        if cuda and rank != -1:
        model = DDP(model, device_ids=[opt.local_rank],
output_device=opt.local_rank,
                        # nn.MultiheadAttention incompatibility with DDP
https://github.com/pytorch/pytorch/issues/26698
                         find_unused_parameters=any(isinstance(layer,
nn.MultiheadAttention) for layer in model.modules()))

        # Model parameters
        hyp['box'] *= 3. / nl  # scale to layers
        hyp['cls'] *= nc / 80. * 3. / nl  # scale to classes and layers
        hyp['obj'] *= (imgsz / 640) ** 2 * 3. / nl  # scale to image size and
layers
        hyp['label_smoothing'] = opt.label_smoothing
        model.nc = nc  # attach number of classes to model



        model.hyp = hyp  # attach hyperparameters to model
        model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
        model.class_weights = labels_to_class_weights(dataset.labels,
nc).to(device) * nc  # attach class weights
        model.names = names

        # Start training
        t0 = time.time()
        nw = max(round(hyp['warmup_epochs'] * nb), 1000)  # number of warmup
iterations, max(3 epochs, 1k iterations)
        # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of
training
```

```
        maps = np.zeros(nc)  # mAP per class
        results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, 
obj, cls)
        scheduler.last_epoch = start_epoch - 1  # do not move
        scaler = amp.GradScaler(enabled=cuda)
        compute_loss = ComputeLoss(model)  # init loss class
        logger.info(f'Image sizes {imgsz} train, {imgsz_test} test\n'
                f'Using {dataloader.num_workers} dataloader workers\n'
                f'Logging results to {save_dir}\n'
                f'Starting training for {epochs} epochs...')
        for epoch in range(start_epoch, epochs):  # epoch -------------------------
-----------------------------------------
        model.train()

        # Update image weights (optional)
        if opt.image_weights:
                # Generate indices
                if rank in [-1, 0]:
                cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # 
class weights
                iw = labels_to_image_weights(dataset.labels, nc=nc, 
class_weights=cw)  # image weights
                dataset.indices = random.choices(range(dataset.n), weights=iw, 
k=dataset.n)  # rand weighted idx
                # Broadcast if DDP
                if rank != -1:
                indices = (torch.tensor(dataset.indices) if rank == 0 else 
torch.zeros(dataset.n)).int()
                dist.broadcast(indices, 0)
                if rank != 0:
                        dataset.indices = indices.cpu().numpy()

        # Update mosaic border
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
        # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

        mloss = torch.zeros(4, device=device)  # mean losses
        if rank != -1:
                dataloader.sampler.set_epoch(epoch)
        pbar = enumerate(dataloader)
        logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 
'total', 'labels', 'img_size'))
        if rank in [-1, 0]:


                pbar = tqdm(pbar, total=nb)  # progress bar
        optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar:  # batch -------------------------
-------------------------------------
                ni = i + nb * epoch  # number integrated batches (since train start)
                imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to 
float32, 0-255 to 0.0-1.0

                # Warmup
                if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio 
(obj_loss = 1.0 or iou)
```

```
                accumulate = max(1, np.interp(ni, xi, [1, nbs /
        total_batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
                        # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to
lr0
                         x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2
else 0.0, x['initial_lr'] * lf(epoch)])
                        if 'momentum' in x:
                                x['momentum'] = np.interp(ni, xi,
[hyp['warmup_momentum'], hyp['momentum']])

                # Multi-scale
                if opt.multi_scale:
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  #
size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                        ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  #
new shape (stretched to gs-multiple)
                        imgs = F.interpolate(imgs, size=ns, mode='bilinear',
align_corners=False)

                # Forward
                with amp.autocast(enabled=cuda):
                pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred, targets.to(device))  # loss
scaled by batch_size
                if rank != -1:
                        loss *= opt.world_size  # gradient averaged between devices in
DDP mode
                if opt.quad:
                        loss *= 4.

                # Backward
                scaler.scale(loss).backward()

                # Optimize
                if ni % accumulate == 0:
                scaler.step(optimizer)  # optimizer.step
                scaler.update()
                optimizer.zero_grad()
                if ema:
                        ema.update(model)

                # Print
                if rank in [-1, 0]:
                  mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
                mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9 if
torch.cuda.is_available() else 0)  # (GB)
                s = ('%10s' * 2 + '%10.4g' * 6) % (
                        '%g/%g' % (epoch, epochs - 1), mem, *mloss, targets.shape[0],
imgs.shape[-1])
                pbar.set_description(s)

                # Plot
                if plots and ni < 3:
                        f = save_dir / f'train_batch{ni}.jpg'  # filename
                        Thread(target=plot_images, args=(imgs, targets, paths, f),
daemon=True).start()
```

```
                    # if tb_writer:
                    #     tb_writer.add_image(f, result, dataformats='HWC',
global_step=epoch)
                     #     tb_writer.add_graph(torch.jit.trace(model, imgs,
strict=False), [])  # add model graph
            elif plots and ni == 10 and wandb_logger.wandb:
                    wandb_logger.log({"Mosaics": [wandb_logger.wandb.Image(str(x),
caption=x.name) for x in
                                                save_dir.glob('train*.jpg') if
x.exists()]})

            # end batch ----------------------------------------------------------
----------------------------------------
        # end epoch ----------------------------------------------------------------
--------------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups]  # for tensorboard
        scheduler.step()

        # DDP process 0 or single-GPU
        if rank in [-1, 0]:
            # mAP
            ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names',
'stride', 'class_weights'])
            final_epoch = epoch + 1 == epochs
            if not opt.notest or final_epoch:  # Calculate mAP
            wandb_logger.current_epoch = epoch + 1
            results, maps, times = test.test(data_dict,
                                              batch_size=batch_size * 2,
                                             imgsz=imgsz_test,
                                             model=ema.ema,
                                             single_cls=opt.single_cls,
                                             dataloader=testloader,
                                             save_dir=save_dir,
                                             verbose=nc < 50 and final_epoch,
                                              plots=plots and final_epoch,
                                             wandb_logger=wandb_logger,
                                             compute_loss=compute_loss,

                                              is_coco=is_coco)

            # Write
            with open(results_file, 'a') as f:
            f.write(s + '%10.4g' * 7 % results + '\n')  # append metrics,
val_loss
            if len(opt.name) and opt.bucket:
            os.system('gsutil cp %s gs://%s/results/results%s.txt' %
(results_file, opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_loss',  #
train loss
                    'metrics/precision', 'metrics/recall', 'metrics/mAP_0.5',
'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',  # val loss
                    'x/lr0', 'x/lr1', 'x/lr2']  # params
            for x, tag in zip(list(mloss[:-1]) + list(results) + lr, tags):
            if tb_writer:
                tb_writer.add_scalar(tag, x, epoch)  # tensorboard
            if wandb_logger.wandb:
                wandb_logger.log({tag: x})  # W&B

            # Update best mAP
            fi = fitness(np.array(results).reshape(1, -1))  # weighted
combination of [P, R, mAP@.5, mAP@.5-.95]
            if fi > best_fitness:
            best_fitness = fi
            wandb_logger.end_epoch(best_result=best_fitness == fi)

            # Save model
            if (not opt.nosave) or (final_epoch and not opt.evolve):  # if save
            ckpt = {'epoch': epoch,
                    'best_fitness': best_fitness,
                    'training_results': results_file.read_text(),
                    'model': deepcopy(model.module if is_parallel(model)
else model).half(),
                    'ema': deepcopy(ema.ema).half(),
                    'updates': ema.updates,
                    'optimizer': optimizer.state_dict(),
                    'wandb_id': wandb_logger.wandb_run.id if
wandb_logger.wandb else None}

            # Save last, best and delete
              torch.save(ckpt, last)
            if best_fitness == fi:
                torch.save(ckpt, best)
            if wandb_logger.wandb:
                if ((epoch + 1) % opt.save_period == 0 and not final_epoch) and
opt.save_period != -1:
                    wandb_logger.log_model(
                    last.parent, opt, epoch, fi, best_model=best_fitness ==
fi)
            del ckpt

        # end epoch ----------------------------------------------------------
--------------------------------------
        # end training
        if rank in [-1, 0]:
        # Plots
        if plots:
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb_logger.wandb:
            files = ['results.png', 'confusion_matrix.png', *[f'{x}_curve.png'
for x in ('F1', 'PR', 'P', 'R')]]
            wandb_logger.log({"Results": [wandb_logger.wandb.Image(str(save_dir /
f), caption=f) for f in files
                                            if (save_dir / f).exists()]})
        # Test best.pt
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start_epoch +
1, (time.time() - t0) / 3600))
        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO
            for m in (last, best) if best.exists() else (last):  # speed, mAP
tests
            results, _, _ = test.test(opt.data,
                                      batch_size=batch_size * 2,
                                      imgsz=imgsz_test,
                                      conf_thres=0.001,
                                                      iou_thres=0.7,
                                                      model=attempt_load(m, device).half(),
                                                      single_cls=opt.single_cls,
                                                      dataloader=testloader,
                                                      save_dir=save_dir,
                                                      save_json=True,
                                                      plots=False,
                                                      is_coco=is_coco)

        # Strip optimizers
        final = best if best.exists() else last  # final model
        for f in last, best:
              if f.exists():
              strip_optimizer(f)  # strip optimizers
        if opt.bucket:
              os.system(f'gsutil cp {final} gs://{opt.bucket}/weights')  # upload
        if wandb_logger.wandb and not opt.evolve:  # Log the stripped model
             wandb_logger.wandb.log_artifact(str(final), type='model',
                                                        name='run_' +
wandb_logger.wandb_run.id + '_model',
                                                        aliases=['last', 'best', 'stripped'])
        wandb_logger.finish_run()
        else:
        dist.destroy_process_group()
        torch.cuda.empty_cache()
        return results


if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--weights', type=str, default='yolov5s.pt',
help='initial weights path')

        parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
        parser.add_argument('--data', type=str, default='data/coco128.yaml',
help='data.yaml path')
        parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml',
help='hyperparameters path')
        parser.add_argument('--epochs', type=int, default=300)
        parser.add_argument('--batch-size', type=int, default=16, help='total batch
size for all GPUs')
        parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640],
help='[train, test] image sizes')
        parser.add_argument('--rect', action='store_true', help='rectangular
training')
        parser.add_argument('--resume', nargs='?', const=True, default=False,
help='resume most recent training')
        parser.add_argument('--nosave', action='store_true', help='only save final
checkpoint')
        parser.add_argument('--notest', action='store_true', help='only test final
epoch')
        parser.add_argument('--noautoanchor', action='store_true', help='disable
autoanchor check')
        parser.add_argument('--evolve', action='store_true', help='evolve
hyperparameters')
        parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
        parser.add_argument('--cache-images', action='store_true', help='cache
images for faster training')
```

```
        parser.add_argument('--image-weights', action='store_true', help='use
weighted image selection for training')
        parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
0,1,2,3 or cpu')
        parser.add_argument('--multi-scale', action='store_true', help='vary img-
size +/- 50%%')
        parser.add_argument('--single-cls', action='store_true', help='train multi-
class data as single-class')
        parser.add_argument('--adam', action='store_true', help='use
torch.optim.Adam() optimizer')
        parser.add_argument('--sync-bn', action='store_true', help='use
SyncBatchNorm, only available in DDP mode')
        parser.add_argument('--local_rank', type=int, default=-1, help='DDP
parameter, do not modify')
        parser.add_argument('--workers', type=int, default=8, help='maximum number
of dataloader workers')
        parser.add_argument('--project', default='runs/train', help='save to
project/name')
        parser.add_argument('--entity', default=None, help='W&B entity')
        parser.add_argument('--name', default='exp', help='save to project/name')
        parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
        parser.add_argument('--quad', action='store_true', help='quad dataloader')
        parser.add_argument('--linear-lr', action='store_true', help='linear LR')
        parser.add_argument('--label-smoothing', type=float, default=0.0,
help='Label smoothing epsilon')
        parser.add_argument('--upload_dataset', action='store_true', help='Upload
dataset as W&B artifact table')
        parser.add_argument('--bbox_interval', type=int, default=-1, help='Set
bounding-box image logging interval for W&B')


        parser.add_argument('--save_period', type=int, default=-1, help='Log model
after every "save_period" epoch')
        parser.add_argument('--artifact_alias', type=str, default="latest",
help='version of dataset artifact to be used')
        opt = parser.parse_args()

        # Set DDP variables
        opt.world_size = int(os.environ['WORLD_SIZE']) if 'WORLD_SIZE' in
os.environ else 1
        opt.global_rank = int(os.environ['RANK']) if 'RANK' in os.environ else -1
        set_logging(opt.global_rank)
        if opt.global_rank in [-1, 0]:
        check_git_status()
        check_requirements(exclude=('pycocotools', 'thop'))

        # Resume
        wandb_run = check_wandb_resume(opt)
        if opt.resume and not wandb_run:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  #
specified or most recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        apriori = opt.global_rank, opt.local_rank
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:
                opt = argparse.Namespace(**yaml.safe_load(f))  # replace
        opt.cfg, opt.weights, opt.resume, opt.batch_size, opt.global_rank,
opt.local_rank = \
            '', ckpt, True, opt.total_batch_size, *apriori  # reinstate
        logger.info('Resuming training from %s' % ckpt)
        else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else
'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_file(opt.cfg),
check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be
specified'
        opt.img_size.extend([opt.img_size[-1]] * (2 - len(opt.img_size)))  # extend
to 2 sizes (train, test)
        opt.name = 'evolve' if opt.evolve else opt.name
        opt.save_dir = str(increment_path(Path(opt.project) / opt.name,
exist_ok=opt.exist_ok | opt.evolve))

        # DDP mode
        opt.total_batch_size = opt.batch_size
        device = select_device(opt.device, batch_size=opt.batch_size)
        if opt.local_rank != -1:
        assert torch.cuda.device_count() > opt.local_rank
        torch.cuda.set_device(opt.local_rank)
        device = torch.device('cuda', opt.local_rank)
        dist.init_process_group(backend='nccl', init_method='env://')  #
distributed backend
        assert opt.batch_size % opt.world_size == 0, '--batch-size must be multiple
of CUDA device count'
        opt.batch_size = opt.total_batch_size // opt.world_size

        # Hyperparameters
        with open(opt.hyp) as f:

        hyp = yaml.safe_load(f)  # load hyps

        # Train
        logger.info(opt)
        if not opt.evolve:
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
            prefix = colorstr('tensorboard: ')
            logger.info(f"{prefix}Start with 'tensorboard --logdir
{opt.project}', view at http://localhost:6006/")
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer)

        # Evolve hyperparameters (optional)
        else:
        # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit,
upper_limit)
              meta = {'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2,
Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
                 'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
                'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
                'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
                'box': (1, 0.02, 0.2),  # box loss gain
                'cls': (1, 0.2, 4.0),  # cls loss gain
                'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
                'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
                'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
                'iou_t': (0, 0.1, 0.7),  # IoU training threshold
                'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
                'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
                'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default
gamma=1.5)
                'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
                'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation
(fraction)
                'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
                'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
                'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
                'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
                'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
                'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction),
range 0-0.001
                'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
                'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
                'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
                'mixup': (1, 0.0, 1.0)}  # image mixup (probability)

        assert opt.local_rank == -1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable
indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save best result
here

        if opt.bucket:
                os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # download
evolve.txt if exists

        for _ in range(300):  # generations to evolve
                if Path('evolve.txt').exists():  # if evolve.txt exists: select best
hyps and mutate
                # Select parent(s)
                parent = 'single'  # parent selection method: 'single' or 'weighted'
                x = np.loadtxt('evolve.txt', ndmin=2)
                n = min(5, len(x))  # number of previous results to consider
                x = x[np.argsort(-fitness(x))][:n]  # top n mutations
                w = fitness(x) - fitness(x).min()  # weights
                if parent == 'single' or len(x) == 1:
                        # x = x[random.randint(0, n - 1)]  # random selection
                        x = x[random.choices(range(n), weights=w)[0]]  # weighted
selection
                elif parent == 'weighted':
                        x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted
combination

                # Mutate
                 mp, s = 0.8, 0.2  # mutation probability, sigma
                npr = np.random
                npr.seed(int(time.time()))
                g = np.array([x[0] for x in meta.values()])  # gains 0-1
                ng = len(meta)
                v = np.ones(ng)
                while all(v == 1):  # mutate until a change occurs (prevent
duplicates)
```

```
                    v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() *
s + 1).clip(0.3, 3.0)
                for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
                    hyp[k] = float(x[i + 7] * v[i])  # mutate

                # Constrain to limits
                for k, v in meta.items():
                hyp[k] = max(hyp[k], v[1])  # lower limit
                     hyp[k] = min(hyp[k], v[2])  # upper limit
                hyp[k] = round(hyp[k], 5)  # significant digits

                # Train mutation
                results = train(hyp.copy(), opt, device)

                # Write mutation results
                print_mutation(hyp.copy(), results, yaml_file, opt.bucket)

            # Plot results
            plot_evolution(yaml_file)
            print(f'Hyperparameter evolution complete. Best results saved as:
{yaml_file}\n'
                f'Command to train a new model with these hyperparameters: $ python
train.py --hyp {yaml_file}')
```