Deep Learning based Forest Fire Detection from UAV Images

Design Project II

July 31st, 2020

University of Victoria

Group ID: 51

Faculty Supervisor: Dr. Xiaodai Dong

Co-supervisor (if any):

Team Members (include all) Information:

#	Name	V Number	Program
1	Devlyn Dorfer	V00846516	Software
2	Matthew Fortier	V00701675	Software
3	Kian Gorgichuk	V00855771	Software
4	James Leahy	V00879166	Software
5	Goh Sato	V00850584	Software

Acknowledgment

We would like to express our deep gratitude to our project supervisor, Dr. Xiaodai Dong, for dedicating her time in guiding us through this project. Her involvement was a major factor in the success of the project. We would also like to thank the SENG 499 teaching team, Dr. Fayez Gebali and our teaching assistant Sanaul Haque, for managing this course. Additionally, we thank the Software Engineering Director, Dr. Stephen W. Neville, for providing the funding for this project through the University of Victoria. Finally, we would like to thank all the people who made their wildfire images and videos publicly available on the internet. Without this data, we would have no method to train or test our machine learning models.

Executive Summary

Every year, governments in Canada spend an excess $800 million in fighting wildfires. These wildfire fighting costs continue to grow annually as climate change increases the overall risk for wildfires in Canada. In order to keep wildfire fighting costs from growing exponentially, early detection of wildfires is required. Unmanned aerial vehicles (UAVs) provide an opportunity to detect wildfires autonomously and survey large amounts of forest without human intervention. Therefore, the goal of the project was to develop a machine learning model that can accurately identify wildfires using images taken from a UAV. The machine learning model would then be executed on an embedded machine learning platform such that the UAV is executing the machine learning model in real- time.

The embedded machine learning platform that was chosen was the NVIDIA Jetson Nano because of its low cost and support for popular machine learning packages such as TensorFlow and Keras. On the NVIDIA Jetson Nano, a total of 6 models were developed using transfer learning. Transfer learning is the machine learning process where a model trained on one set of data is re-trained on another set of data. The models that were developed during the course of the project are: a simple neural net, ResNet50, DenseNet121, MobileNet, and two types of ensemble models (combinations of previous models). Testing of the models were completed and both ResNet50 and DenseNet121 satisfied the design requirements of an accuracy greater than 90% and less than 1 second inference time. Overall, it is recommended that ResNet50 be used as the machine learning model for the UAV because of its consistency in training when compared to DenseNet121 and its shorter inference time.

Table of Contents

List of Figures

List of Tables

Glossary

Base model	In the context of this report, a base model is a pre-trained CNN used to extract features from an image. It is supplemented with a classifier in order to infer image class.
CNN	Convolutional Neural Network, a type of machine learning algorithms commonly used in image classification.
Convolution	A mathematical operation wherein two input functions are integrated together to produce a third function expressing a relationship between their shapes. Convolution is used extensively for image recognition in deep learning.
CUDA	Compute Unified Device Architecture. An architecture developed for parallelized computing on NVIDIA GPUs.
Deep Learning	A biologically inspired subfield of machine learning. Deep learning uses layers of interconnected artificial neurons to simulate the way the human brain functions.
DR	A design requirement of the project.
Embedded System	A component of a larger system where the component has integrated hardware and software.
Keras	A framework built on top of Tensorflow. Provides an intuitive API for programmers to rapidly develop and test deep learning models. Also assists in gathering performance metrics.
LSTM	Long short-term memory. A type of recurrent neural network used to process video and time series data.
Machine Learning	A subset of computer science that focuses on algorithms that improve automatically through experience. Machine learning algorithms commonly have a training phase (where the algorithm is improved) and a testing phase.

NVIDIA Jetson Nano	An embedded machine learning platform.
OpenCV	An open source Python library that focuses on real-time computer vision applications.
Precision	A machine learning metric on the quality or exactness of the classifications.
px	A pixel (measurement of image resolution).
Python	A programming language.
Recall	A machine learning metric on the probability that the model will correctly classify a true positive.
TensorFlow	A python library facilitating the creation of machine learning models. TensorFlow is used extensively for deep learning.
TensorRT	A software development kit used to optimize inference performance of deep neural networks.
Transfer Learning	Transfer learning is the process in machine learning where a machine learning algorithm is trained in one problem and is then modified/retrained to solve a different but related problem.
UAV	Unmanned Aerial Vehicle.

1.0 Introduction

In Canada it is estimated that Canadian Wildland fire protection agencies have spent between $800 million and $1.4 billion in managing wildfires [1]. These costs for fighting wildfires continue to rise, as climate change continues to warm the planet [1]. For these reasons, early detection of wildfires is vital to keeping the costs of fighting wildfires from growing exponentially. The earlier a wildfire is detected, the smaller the wildfire is to contain which results in lower overall costs.

The main methods of detecting wildfires rely on a human determining if a wildfire is occurring in a specific area. The detection may occur by wildfire fighting personnel or the general public. The problem with human detection is that it is impossible to have wildfire detection running 24 hours a day, 7 days a week without significant cost. That being said, autonomous UAVs provide the ability to have 24/7 wildfire monitoring and detection while observing significantly more forest area than human observers.

The group chose this project because the project members are all interested in working on a machine learning project and the domain of wildfire sensing has a high potential for societal impact. Wildfires decimate communities and cost governments millions of dollars each year [1]. The end goal of this project is to allow government forest fire programs to detect and extinguish forest fires earlier, thus saving money on wildfire fighting costs. Therefore, the potential user base would be government wildfire programs.

2.0 Project Goal

The goal of this project is to research and develop a machine learning model that can accurately detect wildfires from images taken from a UAV. This machine learning model will then be installed on an embedded machine learning platform so that the UAV itself can detect the wildfires in real-time, creating an early warning detection system for wildfires that is completely autonomous. Note that the scope of this project is the machine learning model itself and not other systems of the UAV such as flight control or navigation.

3.0 Design Objectives

There are two design objectives and three design constraints for the project. The first design objective is the machine learning model has an accuracy greater than or equal to 90% when classifying a set of data. Obviously, the machine learning model should have an accuracy as close to 100% accuracy as possible however 90% accuracy is the minimum threshold. The 90% accuracy threshold was chosen to ensure that the model produces accurate results without too many errors. The second design objective is that the model must produce a prediction for an

image within 1 second. A value of 1 second was chosen because the machine learning model will be executed on the UAV. Assuming the top speed of the UAV is 100 km/h, then the UAV will be moving approximately 28 m/s. Thus, the UAV can determine a wildfire from an image every 28 meters when moving at top speed.

The first design constraint is that the model must work on images of a minimum resolution of 720p (1280px by 720px). A minimum resolution is required to provide sufficient detail in order for the model to detect the wildfires. The second design constraint is the machine learning model must execute on an embedded learning platform that is mounted on the UAV. The reason for this constraint is because of the project requirements for early wildfire detection. Finally, the third design constraint is the model must have a pre-processing mechanism to extract video frames from videos and a video stream. This design constraint is because the machine model will be executing on the UAV and thus will have the UAV video feed being sent to the machine learning model for classification of images.

4.0 Literature Survey

The literature review for the project had two main goals. The first goal was to identify potential machine learning approaches to attempt and compare results. The second goal was to gather images and videos of forests from an aerial view (example: taken from a drone, helicopter, or airplane). These images and videos include forests, plains, visible wildfires, and smoke from wildfires. In total over 20 academic articles were reviewed for this project. Table 1 contains all the academic articles that were found to contain useful information and provide insights to accomplish this project. Table2 contains the academic articles that are in the same domain as the project but are not relevant to the project specifications.

Table 1. Literature with useful information and insights.

Literature	**Useful Information and Insights**
A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques [2]	An overview of the forest fire detection problem space. Discusses the validity of differing methods of detection, such as infrared vs visible light detection, smoke vs flame detection, and classic computer vision techniques vs deep neural networks. Also contains insights on implementation concerns such as image stabilization to account for UAV vibration, and other important attributes such as geolocation and prognosis of detected fires.
Saliency Detection and Deep Learning-Based Wildfire Identification in UAV Imagery [3]	A study and field-test of a deep learning system for forest fire detection. The authors used a convolutional neural network for classification, and placed a heavy emphasis on image preprocessing. The authors note that the greatest challenges in this application are the lack of acceptable training datasets, low image resolution for live classification, and the inherent difficulty of classifying objects without clear contours.
Convolutional Neural Networks Based FireDetection in Surveillance Videos [4]	Another example of a neural network-based fire detection system. The authors use Google's GoogLeNet architecture, but retrain the model from randomized weights to achieve a highly accurate fire detection system with a low computational cost at runtime. The fires being detected in this case are not specifically forest fires. Instead, the focus is on general fire detection in a surveillance context. The result was a convolutional neural net with high recall, perhaps at the cost of precision. Overall efficacy was higher than existing feature-based machine learning models.

A Deep Learning Based Forest Fire Detection Approach Using UAV and YOLOv3 [5]	The authors of this paper use the real-time object based detection system YOLOv3 to detect wildfires. The authors accomplished this by adding additional layers to the YOLOv3 model. This resulted in an accuracy of 82% at 3.2 frames per second. The authors also noted that they struggled to find enough relevant data to test their model.
Vision-based Forest Fire Detection in Aerial Images for Firefighting Using UAVs [6]	The authors of this paper propose a technique where images are transformed into binary images, then fire zones are identified based on pixel connectivity and interest areas. The height and width of the firezones are then determined along with the firezones' position. All of this machine learning processing happens in a ground station which the UAV transfers images to using a 5.8GHz 200mW transmitter and a 5:8G AV receiver. It should be noted that this system requires that the UAV hover to confirm the detection of a wildfire, therefore a fixed wing UAV would not be suitable for this approach.
Wildfire Smoke Detection Using SpatioTemporal Bag-of-Features of Smoke [7]	The authors develop a smoke detection system for ground station mounted cameras. The authors chose to focus on smoke detection because smoke is an early indicator of fire when compared to training a model that identifies flames. The authors compared optical based solutions (with infrared and LIDAR sensors) to a charge-coupled device (CCD) camera (color). They determined that CCD cameras are better for wildfire smoke detection because optical based solutions create false alarms because of light reflection, the conditions of the weather and the distance between sensors and the fire. The machine learning approach the authors used was a random forest classifier, which is an ensemble of decision trees. This was built during the training phase by using the BoF histogram.

Recurrent convolutional network for video-based smoke detection [8]	The authors for this paper focus on detecting smoke and not visible fire. Note that this paper is not specific to wildfires but any type of smoke in general. The authors document the difficulty in finding relevant datasets for smoke detection. The authors were able to provide some examples of datasets used by other academic papers however, they resorted to building their own dataset because the datasets available were insufficient for their objectives. The method they used to gather their dataset was by using a search engine with specific keywords to download videos from the internet. The authors concluded that smoke detection only works if a sufficient amount of smoke exists. If the amount of smoke is low, this will correspond with a low detection rate because the area covered by the smoke is relatively small and is vulnerable to interference.
Dual Deep Learning Model for Image Based Smoke Detection [9]	The authors use two Convolutional Neural Networks (CNNs) to detect smoke. One CNN uses feature detection with transfer learning from AlexNet, while the other detects motion based features. Additionally, the authors employ the use of super pixel segmentation to augment their dataset. The resultant model has strong performance with accuracies in the high nineties in both close and far away smoke detection. Moreover, the dual CNN approach allows strong performance with accuracy in the low nineties in challenging environments such as fog or sandstorms where other approaches often fail. However, due to the use of motion information this approach is not viable as a fixed wing UAV moves continuously.
An Attention Enhanced Bidirectional LSTM for Early Forest Fire Smoke Recognition [10]	The authors achieved 97.8% accuracy with 4.4% improvement over image-based deep learning model by using an LSTM (Long Short-Term Memory) machine learning model. The model focused on smoke detection by using the motion of the smoke which requires a still camera. The dataset that was used by this paper was a selection of videos given by the Nanking Enbo Technology Company.

Computer vision for wildfire research: an evolving image dataset for processing and analysis [11]	Researchers found that there was no large public database for wildland fire images and the vast majority of research uses Internet collected images or datasets of fire images non-available publicly. Therefore, the researchers developed their own dataset of about 200 images. Finally, the paper briefly discusses machine learning methods and pixel analysis using the dataset but the main purpose of the paper is the creation of the dataset.

Table 2. Literature in the same domain but not relevant to project specifications.

Literature	**Summary**
Land Use Classification in Remote Sensing Images by Convolutional Neural Networks [12]	An overview of a CNN model for classification of land use based on satellite imagery. While the authors' application is not the same as the project, the article makes a compelling argument for the use of transfer learning to achieve high accuracy CNNs at a low cost.
Convolutional Neural Network Based Automatic Object Detection on Aerial Images [13]	Similar to the previous literature, the authors use transfer learning to reuse an existing CNN architecture for aerial object classification. Input image resolution is much higher (5000x5000 pixels), resulting in long classification times. The authors achieved high accuracy with relatively low training time.
Fire Detection Using Infrared Images for UAV-based Forest Fire Surveillance [14]	This paper focuses on using infrared images for wildfire detection as opposed to color images. While the authors of this paper do not cover a machine learning approach, it does provide useful insights of histogram-based segmentation for image recognition. This type of algorithm could be used to help transform the data before being fed into a neural net.
Deep neural networks for wildfire detection with unmanned aerial vehicle [15]	An extremely short paper that did not provide any useful machine learning approaches. However, this paper does provide more evidence that the dataset collection is a significant problem for developing a model. The authors for this paper used images found using the Google Image search engine. These images were then cropped by hand.

Fire Detection Using Both Infrared and Visual Images With Application to Unmanned Aerial Vehicle Forest Fire Surveillance [16]	The authors use combination images of visible light and infrared, with the infrared image being added as a layer similar to a chrominance layer. This improved accuracy over either method in isolation but required more computational overhead.
Early Forest Fire Detection Using Drones and AI [17]	A high level paper that introduces neural networks in abstract concepts. Discusses using Tensorflow's Object-Detection API to automate the design of the neural network. Appears to be describing a basic feed-forward network rather than a convolutional one. Also provides significant detail on the deployment of the UAVs.
Computer vision system for fire detection and report using UAVs [18]	The authors of this paper focus on the hardware and software controlling the UAV. They briefly discuss fire detection using OpenCV and Numpy but the only testing executed was by burning newspapers and not on actual wildfire images.
Early forest fire detection by vision-enabled wireless sensor networks [19]	The main idea of this paper is a distributed network of cameras that each do image processing to detect wildfires. The benefits of this type of distributed network is it is easier to scale as bandwidth and power consumption become less of a concern in a distributed setting. Another goal of this paper was to keep power consumption to a minimum as they were targeting IoT devices As it is a distributed network of cameras, the machine learning aspects of the paper focus on stationary cameras. However, the authors had good results given the power and performance constraints. This paper would be a good reference for optimizing the machine learning algorithm.

Throughout the literature review it became apparent that acquiring a dataset with sorted aerial images of wildfires and aerial images of forest would be a major issue. Without enough correctly labelled images a machine learning model cannot be trained properly. Therefore, a significant portion of the literature review focused on acquiring datasets from academic sources and websites on the internet. The datasets found and used can be seen in section 6.3's table 4.

The literature review also identified that there were many different approaches to detect wildfires using machine learning:

- Smoke detection vs. fire detection vs. combination of both.
- Models that relied on stationary cameras vs. models that do not.
- CNN model architecture vs. LSTM model architecture.

It should also be noted that a significant amount of machine learning literature involving wildfires smoke detection relies on the movement of the smoke. The reason for this is because fog can easily be confused as smoke by machine learning models, thus by tracking the movement of the smoke, this avoids false positives generated by fog. However, this can only be done using stationary cameras. Given the requirements and constraints of the project, no model could rely on stationary cameras as cameras mounted on the UAV are not stationary. If the UAV had to stop and cover to make a wildfire prediction this would reduce the amount of land that the UAV could cover. That being said, all other approaches are discussed in section 6.0 Design Methodology and Analysis.

5.0 Team Duties & Project Planning

The project had many integrated components that required close collaboration between all five team members. Thus, the concept of a manager was used to assign responsibility for project work. Note that a manager in the context of the project did not complete all assigned project work by themselves. All members of the team contributed to all aspects of the project, a manager simply signifies responsibility to ensure that the work is completed by the team. Table 3 details the role assignments for the project.

Table 3. Role assignments.

Roles	**Assigned**
Project Manager	Kian Gorgichuk
Data Manager	Goh Sato
Model Manager	Devlyn Dorfer and Matthew Fortier
Embedded Manager	James Leahy

5.1 Project Manager

The project manager is responsible for coordinating the team and the success of the overall project. The main deliverable assigned to the project manager is ensuring the documentation (final report, poster, and video) of the project is accurate and complete. Note this position was originally called project coordinator in the progress report. The responsibilities that the project manager has includes:

- Arranging and leading all group meetings.
- Facilitating communication with the project's supervisor and the SENG 499. teaching team.
- Determining and assigning work to team members on a weekly basis.
- Reviewing and submitting all deliverables.
- Ensuring that communication between different team members.

The project manager also provides additional assistance when team members need extra resources. For example, the project manager assisted the data manager in collecting training/testing data for the models and the project manager assisted the model managers in prototyping ensemble models.

The main challenge for the project manager is the ongoing pandemic that occurred during the course of the project. The pandemic resulted in all project work and meetings being done remotely, which introduces challenges in communication and teamwork as the team could not meet in person. To solve this issue, multiple status update meetings were arranged and led by the project manager with various team members to guarantee sufficient visibility within the project. The project manager also led the meeting between the team and the project supervisor.

5.2 Data Manager

The data manager was responsible for all machine learning data used in the project. Responsibilities of the data manager included acquiring, organizing, and preprocessing data to be used in the machine learning models.

The main deliverable the data manager is responsible for is the final dataset used for model training and verification. The training dataset is a vital component in the development of the machine learning model and is therefore closely tied to the requirement of the model detecting wildfires with 90% accuracy. The testing dataset should sufficiently emulate real wildfire footage captured by drones to validate that the model is effective. This means the footage must come from an aerial viewpoint (no ground stations or satellite images) and many

videos of different wildfires must be used. Additionally, the dataset must be large enough to ensure the model does not overfit the data.

The main challenge experienced by the data manager was the lack of available dataset with sufficient wildfire images. The literature review showcased that many academic papers used either self generated data or data acquired from the internet. Thus, the team had to develop their own dataset to fit the project requirements.

5.3 Model Manager

The model managers are responsible for the research and development of the machine learning models built during the project. The main deliverable model managers were responsible for was the creation of multiple machine learning models that are capable of detecting wildfires. As the machine learning models are a considerable portion of the project, two team members were assigned to this role to ensure adequate resources for research and development. These models were then compared to determine the machine learning model that best satisfied the design objectives.

The main challenge experienced by the model managers was determining the applicable machine learning methods and techniques that could apply to this project. The field of machine learning has many different methods to accomplish image classification, and these methods have different parameters that can be changed which affect the performance and accuracy of the models. With so many possibilities, the model manager had to ensure that the scope of research and development was not exceeding the resources (time) allocated to the project because there is potentially an infinite number of ways to change and modify machine learning models.

5.4 Embedded Manager

The embedded manager was responsible for all embedded aspects of the project. The main responsibility of the embedded manager is selecting what platform would be attached to the UAV and transferring the machine learning model, created by the model managers, to the embedded machine learning platform. The embedded manager also developed performance metrics and executed performance tests of the machine learning models on the embedded platform.

The main challenge experienced by the embedded manager was transferring the models onto the embedded systems machine learning platform. Incompatibilities with software and architecture of the embedded systems machine learning platform. To mitigate these issues, a

significant amount of research and troubleshooting was done by the embedded manager to understand the incompatibilities of the embedded system.

6.0 Design Methodology & Analysis

This section covers the main milestones of the project: hardware selection, algorithm selection, data gathering, model development and refining, video to image conversion, and transferring the models to the embedded system.

6.1 Selection of Hardware

The NVIDIA Jetson Nano was selected because of the following reasons:

1. Support for a modern machine learning stack via acceptable hardware (GPU) and software (Tensorflow).
2. Fell within the price budget for the project [20].
3. Good development community and resources available such as community forums and guides.

Other hardware platforms were briefly considered, but for the purposes of time efficiency and because of the fact the Jetson Nano satisfied all constraints of our project it was quickly selected. The relevant specifications of the embedded platform are:

1. 128-Core Maxwell GPU
2. Quad-core ARM A57 @ 1.43 GHz
3. 4 GB 64-bit LPDDR4 25.6 GB/s
4. 32 GB MicroSD card

6.2 Algorithm Selection

To build a classifier for wildfire detection, it was crucial to select an appropriate classification method. Many algorithms have been examined and tested in the literature, including wavelet analysis, histogram-based approaches, and neural networks. These approaches have shown various levels of success as noted in a meta analysis by Yuan et al [2].

It was decided to pursue deep neural networks for classification. This decision was driven largely by efficacy, ease of implementation, hardware integration, and domain familiarity. Several frameworks exist to rapidly prototype neural networks with high level API calls, such as Tensorflow and Pytorch. Resulting in more time could be spent on algorithmic design and optimization as opposed to low-level implementation. Additionally, the NVIDIA Jetson chips are

designed with these frameworks in mind, as both can utilize the CUDA cores in the graphics processor for rapid inference [21]. Finally, the team members of the project already had some familiarity with the theoretical foundations of neural networks; many alternative methods such as wavelet analysis required advanced knowledge of signal processing, presenting a high barrier to entry.

Two types of deep neural networks were considered. Convolutional neural networks (CNN) for still image classification, and Long Short-Term Memory (LSTM) networks for video classification. Prior research shows that the movement of smoke or flickering of flames in a video present easy features for classifying forest fires [2]. However, these experiments were offline video classification where computing resources were not tightly constrained. When deploying the model to an embedded system on a UAV, storage and processing of video data could quickly become an intractable problem for nontrivial flight lengths. Any turbulence or vibration during flight may also cause image blurring, requiring stabilization and further compounding resource concerns. For these reasons, video classification was ruled out in favour of still-image classification via a CNN.

6.2.1 A Primer on Convolutional Neural Networks

Convolutional neural networks are a subclass of neural networks used for feature extraction from grid-like data such as images and time series [22]. Specifically, they contain at least one convolution layer where a small matrix (called a kernel) is convolved with the image to create several smaller feature maps. Convolution layers implement a discrete version of convolution where flipping of the kernel is not necessary. This is called cross-correlation [22], and has the following form:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{1}$$

Where S is the feature map, K is the kernel, and I is the input image. In the case of image processing, these are all two-dimensional grids. Each kernel is convolved with each axis in the previous layer (such as the red, green, and blue axes of an image). This means the number of feature maps resulting from a convolution layer can be expressed as:

$$|\mathbf{S}| = |\mathbf{K}| \cdot |\mathbf{I}| \tag{2}$$

Where **S** is the set of feature maps in the output layer, **I** is the set of images in the input layer, and **K** is the set of kernels used. These feature maps can then be reduced in size using

pooling layers. Different downsampling methods can be used for pooling layers, with max pooling and average pooling being common examples. Effective CNNs will contain many convolution and pooling layers, until the feature maps are reduced to singular values (features) which are often fed into a classification network. This process is illustrated in figure 1.

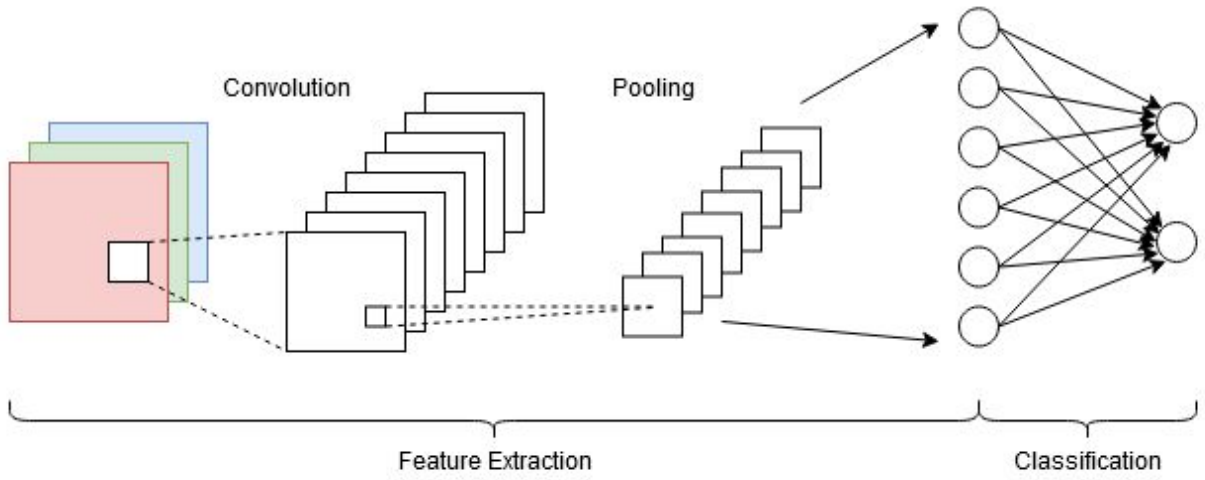

Figure 1: Generic CNN architecture.

6.3 Data Gathering

Data was drawn from a variety of sources to maximize volume and ensure generalizability. Several public data repositories on GitHub were utilized, as well as a private database owned by Corsican University [23]. This was supplemented by over one hundred images scraped from Google image search and extracted from drone video footage obtained from Youtube. A breakdown of data sources is provided in table 4.

Table 4: Datasets used in model development.

Dataset Name	**Type of Data**
Drone image of a fire burning in the forest [24]	Single video from a drone observing a structure fire and smoke within a forest.
Beautiful Drone Flight Over Forest with Pink Floyd Sorrow [25]	Single video from a drone flying over a forest.

DJI Spark flies over forest fire. [26]	Single video of from a drone of a wildfire in the distance and close up.
Corsican Fire Database [23]	600+ images. Several videos in both visible and near infrared spectrums.
Fire-Flame-Dataset [27]	3000+ images divided into fire, smoke, and neutral.
Fire-Detection-Image-Dataset [28]	600+ images, ~120 fire (not forest specific) and the rest are just random non-fire photos.
FIRE Dataset [29]	999 outdoor images of forest. 755 images of forest fire. Non-aerial, ground images.
Aerial Nature Videos, Amazing Drone Footage of Forests and Meadows [30]	YouTube video of drone, aerial forest, no fire.
Planted Forest l Eagle Eye Project l Borneo l Aerial View [31]	YouTube aerial footage of a forest from a drone with no wildfires.
Amazing 4k Drone Video of this Canadian Mountain Town | Flights EP. 01 [32]	5 minutes of aerial drone footage of the Canadian mountains with no wildfires
Drone Footage From Rare Wildfire in Iceland [33]	Approximately 2 minutes of aerial drone footage of a small wildfire.
Yost Ridge Wildfire [34]	1 minutes of wildfire footage in both a field and in a forest.
Aerial view from CTV News chopper shows raging fire in B.C. [35]	2 minutes of footage of a wildfire with visible flames and a significant amount of smoke
Aerial video shows daunting scale of B.C. wildfire | CTV News [36]	20 seconds of a wildfire with visible smoke and fire.

Aerial footage of massive Alberta wildfire | CTV News [37]	Approximately 1 minute of aerial wildfire footage in the Alberta forest.

Data was pruned and organized manually, and duplicate images were removed. Several subsampled datasets were tested for model training due to the wide variety of acquired images. The results were most favourable when urban fire images were removed. This drastically cut down the size of the dataset, as the largest repository used (DeepQuest fire dataset) was primarily urban images.

Data autocorrelation became a significant concern when still images from videos were added to the dataset. Any two images from the same video had a very similar appearance. This has the potential to allow the model to 'cheat' if those images are split between training, testing, and validation sets. To address this, video-extracted images were all placed in the training set to reduce the chance of an overly optimistic test accuracy.

6.3.1 Dataset Video to Image Conversion

While many of the datasets were collections of images, other datasets contained videos. For training of the machine learning model, images needed to be extracted from the videos and categorized. To accomplish this, the ffmpeg tool was used [38]. This tool allows for videos to be converted to a set of images at set intervals. For the purposes of the project, an image was extracted from each video in the dataset every 1 second.

6.4 Model Development

In total 6 models were developed over the course of the project. These models are a ResNet50 based model, a DenseNet121 based model, a MobileNet based model, 2 ensemble models, and a simple neural net model. This section details how these models were built.

6.4.1 Framework Selection

Tensorflow was selected as the framework for model development. Several team members had familiarity with it, and it integrates well with the Keras framework to ease cognitive load on the developer (using the integrated TensorFlow Keras library). Tensorflow is also part of the Jetson Nano's native SDK "JetPack", so there was no need to compile it from the source code. Finally, models saved from Tensorflow can be optimized using the TensorRT library for faster inference times.

6.4.2 Transfer Learning

Transfer learning, the use of a previously trained network as a starting point, was used to aid in the development of the models. Transfer learning decreases the training time required to achieve satisfactory inference, which reduces the overall iteration time. Additionally, the literature suggests transfer learning is highly effective for aerial image classification [39][12].

6.4.2.1 Base Model Selection and Testing

Numerous base models are available in the TensorFlow Keras library. Candidate models were required to be built for image classification rather than image localization. To keep training time to a minimum, the model must be trainable in batch sizes of at least 32 on the hardware available to the team. Finally, they must be supported by the version of TensorFlow available on the NVIDIA Jetson Nano - v2.1.

In particular, ResNet50, DenseNet121, and MobileNet were chosen for their balance of accuracy, efficiency, and size. These CNNs have all been used to great effect in the ImageNet competition for object classification [40] and their underlying architectures are discussed in Appendix A. Each was tested by training a logistic regressor on their respective outputs. All three were able to converge to accuracies over 0.80 in under 5 epochs, indicating they were suitable for further exploration and tuning.

6.4.2.2 Hyperparameter Tuning

Neural networks have many hyperparameters affecting their training and inference performance. Nonstructural hyperparameters addressed were the optimization algorithm and any associated sub-parameters. Structural hyperparameters include the number of hidden layers, the number of neurons per layer, and the activation function used in each layer. These parameters were only modified for the classification portion of the model, as the pretrained base model had already been finely tuned.

Stochastic gradient descent was chosen as the optimization algorithm. Attempts to use other popular optimizers such as Adam resulted in wildly inconsistent validation accuracy between epochs. Learning rate and momentum were left at their default values of 0.01 and 0 respectively. These values produced satisfactory results, and the optimization algorithm was not prioritized for experimentation due to time constraints.

Activation functions were chosen based on the purpose of each layer. The classification layer used softmax activation with two output neurons. Softmax is the multiclass generalization of the sigmoid function for classification. It outputs a probability at each terminal node, with the

sum of all terminal probabilities adding up to 1. Having the networks output a probability was useful for the construction of ensemble models discussed in section 6.4.3. Hidden classification layers used the rectified linear unit (ReLU) activation function, for its demonstrated ability to reduce the vanishing gradient problem and increase the chance of model convergence [41].

Hidden layer size and quantity were chosen through systematic trial and error. Each model was trained with one, two, and three hidden layers containing between thirty and one hundred eighty neurons. Additionally, the models were trained with no hidden layers; this represents a linear combination of the features extracted by the base models. ResNet50 and DenseNet121 both performed best with a single hidden layer of sixty neurons, achieving test accuracies of 0.9348 and 0.9424 respectively. MobileNet performed best with no hidden layers, achieving a test accuracy of 0.8772. Further analysis of these results and the tabular data from testing are presented in Appendix B.

6.4.2.3 Finalized Model Training

After all hyperparameters were selected, a final training run was performed for each model. The ResNet50 model was trained with one hidden classification layer of sixty neurons. The DenseNet121 model was also trained with one hidden classification layer of sixty neurons. The MobileNet model was trained with no hidden layers. Each model was trained for 5 epochs of all 1673 training images, with a validation step of all 524 validation images between each epoch. Validation accuracy and loss showed no significant gains after 5 epochs for all models, so training beyond this point risked overfitting. The loss and accuracy plots from this process are shown in figure 2 and figure 3.

Figure 2. Loss and accuracy during training and validation over five epochs for ResNet50 and DenseNet121.

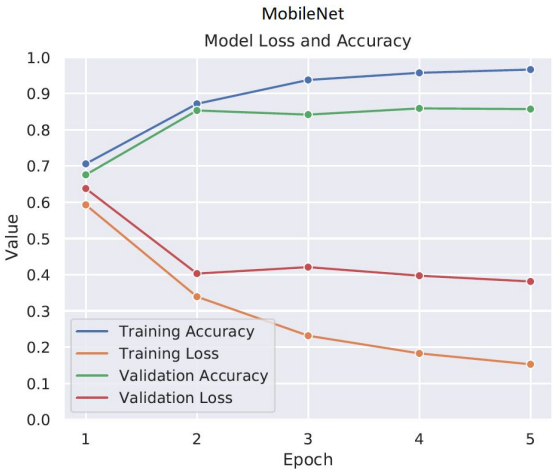

Figure 3: Loss and accuracy during training and validation over five epochs for MobileNet.

6.4.3 Ensemble Learning

In an effort to push classification accuracy as high as possible, two ensemble classifiers were constructed using the three aforementioned models. One ensemble classifier used the majority voting method described by Ju, Bibaut & van der Lan [42]. For each input image, the ensemble model would preprocess and feed the input into each respective model. The models would then vote on the class of the image, with a two-thirds majority required to determine the inference of the ensemble. The other ensemble classifier was similarly designed, but instead of binary voting, the classifier pooled the collective probabilities of the three models. This process is illustrated in figure 4.

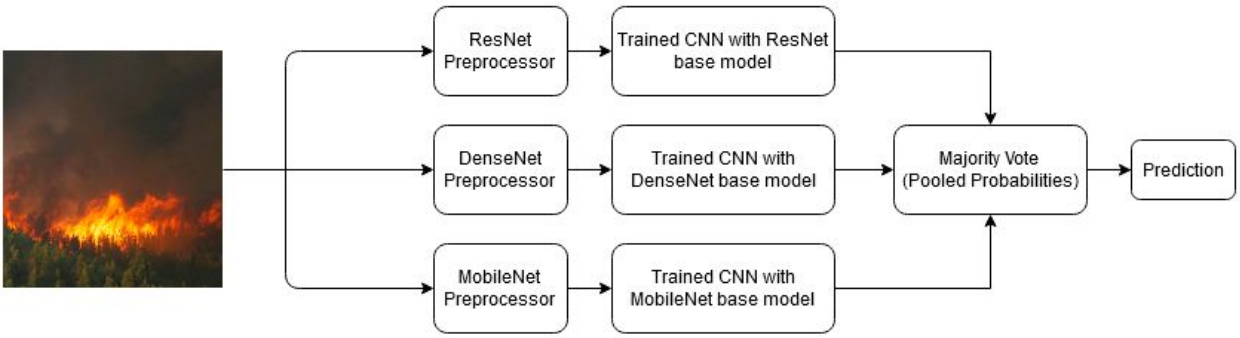

Figure 4: Ensemble classifier architecture.

Another ensemble classifier using probability pooling was constructed using two models based on ResNet50: a smoke detecting binary classification model, and a fire detecting binary classification model. Due to a lack of data, the smoke model is inaccurate and therefore makes the entire ensemble inaccurate.

6.4.4 Simple Neural Net Model

In addition to the transfer learning based models and the ensemble models created from these models, a simple neural net was created to serve a baseline for in terms of the accuracy of all the models that were created during the project. The simple neural net had the following layers:

1. Flattening layer
2. A 25 dimension dense layer with relu activation
3. A 10 dimension dense layer with relu activation
4. A 2 dimension dense layer with softmax activation.

It was anticipated that the simple neural net model would have the worst accuracy of all the models created in the project. The results of testing of the simple neural net model is available in section 7.3 Test Results.

6.5 Video Preprocessing of Data

To enable camera input and video to be processed by the script, a Python script was created. The script works by first loading a previously trained model from an HDF (Hierarchical Data Format) file. Then the script uses OpenCV to sample 4 frames per second from a video. Or when a camera input is specified, it captures 1 frame per second. The script runs inference on each frame to generate a result. The result is overlaid on the frame. The processed frame is displayed on an OpenCV window and saved to a video file. Figure 5 is an example of a single processed frame. It is anticipated that this script will be modified to support the execution of the model on the UAV at a later point in time.

Figure 5: Video processed image.

6.6 Transferring the Model to the Embedded Machine Learning Platform

Tensorflow provides functionality to save model parameters after training. The models get saved in a file formatted as HDF (Hierarchical Data Format) with the extension .h5. These .h5 files are then transferred onto the embedded device, the NVIDIA Jetson Nano, and loaded into memory via a Tensorflow method.

A python script was created which loads models and executes inferences on image data. The inference computation is dispatched to the GPU which is on the embedded system. The dispatching is handled by Tensorflow, but NVIDIA GPU drivers (CUDA) must first be installed on the embedded device.

7.0 Testing & Validation

After all the models were built, a test plan was executed on the models to determine the performance and accuracy of all six models.

7.1 Test Plans

For the purposes of satisfying the design requirements of the project the test plan contains two areas for which test methodologies were put in place. The two areas of interest were off-device and on-device testing where the device is the NVIDIA Jetson Nano.

7.1.1 Off Device Test Plan

Due to the nature of the project much of the testing and development of the model was able to be conducted off device on a more powerful machine than what is available in the production setting. Off device testing focused on measuring and improving four model performance metrics/artifacts: model accuracy, confusion matrix, precision, and recall. The metrics are generated by testing the performance of the models on the testing dataset. The confusion matrices are available in Appendix C.

The accuracy value gives insight into how well the model performs at classifying images. The model confusion matrix gives performance data on the number of true positives (TP), true negatives (TN) , false positives (FP), and false negatives (FN) based on classification of the test data set. Model precision is derived from the confusion matrix and gives a metric on the quality or exactness of the classification. Model recall is derived from the confusion matrix and in the case of binary classification, gives a metric on the probability that the model will correctly classify a true positive.

7.1.2 On Device Test Plan

To satisfy the project design requirements testing was conducted on the production embedded device. The design requirement to be verified on device was the average inference time required to classify an image. The average inference time gives insight into how many images we can classify per unit time.

The hardware platform that the test was run on was the Nvidia Jetson Nano. The software environment consisted of a full linux operating system with various GPU drivers and application software installed (Tensorflow and OpenCV).

7.2 Test Procedure

Code was added to the project in order to automate the collection and testing of both the off device and on device testing.

7.2.1 Off Device Test Procedure

The manner in which the above metrics were collected was implemented in code and setup to be generated upon training a model. Note that the models were tested on a collection of 399 images.

7.2.2 On Device Test Procedure

The saved models (i.e.h5 files) are loaded onto the NVIDIA Jetson Nano developer kit and the average inference time is computed for each model by averaging the time required to classify over 469 images.

7.3 Test Results

Table 5 details test results for each of the models are presented below in addition to whether the model satisfies the design requirements (DR) of the project.

Table 5. Model testing results.

Model	Accuracy	Precision	Recall	Inference Time (s)	Inference Time DR Satisfied	Accuracy DR Satisfied
Simple Neural Net	0.6541	0.6348	0.73	0.1748	YES	NO
ResNet50	0.9348	0.9781	0.895	0.3858	YES	YES
DenseNet121	0.9424	0.9680	0.91	0.3984	YES	YES
MobileNet	0.8772	0.8663	0.875	0.2570	YES	NO
Ensemble (Pooling)	0.8571	0.9689	0.935	1.2447	NO	NO
Ensemble (Voting)	0.8571	0.9687	0.93	1.1520	NO	NO

7.4 Inspection of Misclassified Images

To identify the potential weaknesses of the models, misclassified images were collected for manual inspection. Figure 6 shows a collection of misclassified images from an instance of the ResNet50 model. The file names indicate the truth values of the files (the opposite of what was identified by the model). It can be seen that the presence of direct sunlight can cause the model to misidentify images. However, when the model was restrained and the misclassified images were re-inspected, there was no identifiable overlap or pattern in misclassified images.

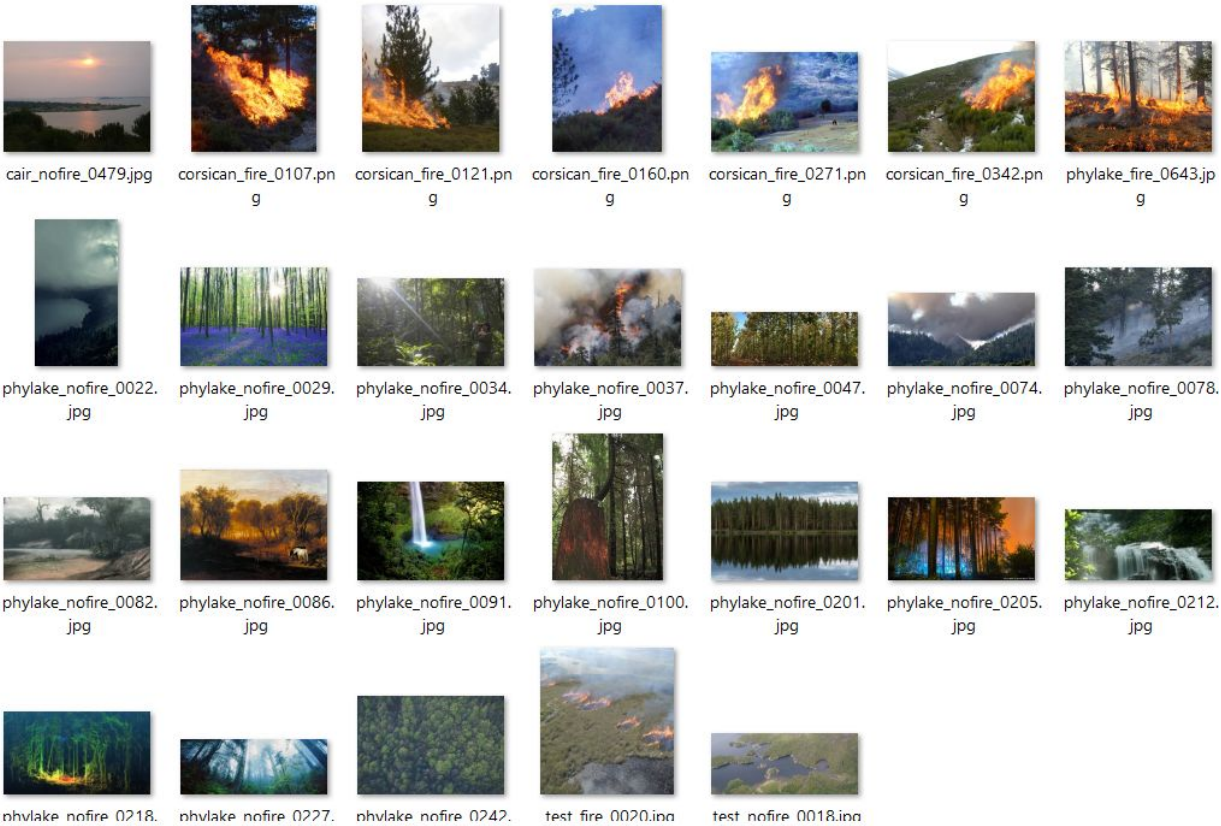

Figure 6. Misclassified ResNet50 images.

8.0 Discussion & Recommendations

Based on the results presented in section 7.3 Test Results, DenseNet121 and ResNet50 are the two models that satisfied both of the project's design requirements. Each having accuracies above 90% and inference times below 1 second.

These results are consistent with what is known in industry as both DenseNet121 and ResNet50 are generally recognized as highly performant models [40]. The simple neural net's performance in terms of accuracy and inference time is expected as it was a small architecture (i.e. failed to meet accuracy requirements, but satisfied inference time requirements). MobileNet is interesting as it appears to be a reasonable balance between accuracy and inference time, as it almost meets the desired accuracy threshold whilst maintaining one of the faster inference times. Finally, the Ensemble Pooling and Ensemble Voting models both did not satisfy the accuracy or inference time requirements. Each model is available publicly on GitHub (https://github.com/devlyn37/DeepFire). That being said, the data used to train and test each model is not available due to web hosting constraints.

Overall, it is recommended that the ResNet50 model be selected as the machine learning model to be used on the UAV. This model showed a higher precision value, making it less likely to set off false alarms and waste valuable human resources. It also showed better classification accuracies on average than any other model, despite not having the highest peak value for our particular testing set. We feel this may lead to better generalization in the field.

Improvements to the existing models created can also be made in multiple areas. Model inference time could be optimized by using TensorRT and by running the production models on a non-developer version of the NVIDIA Jetson Nano. Additionally, the accuracy of the models could be improved to 95%+ with additional datasets and fine tuning of the models.

The next steps for the project would involve defining and integrating the machine learning model (executing on the NVIDIA Jetson Nano) with an actual UAV. This would require defining the hardware and software that the UAV would use, which would include its flight , navigation, and communication software. Additionally, a notification system would also need to be built to notify the relevant people when a wildfire is detected by the UAV.

9.0 Conclusion

Automated wildfire detection using UAVs and machine learning provide an opportunity to build an early detection for wildfires, resulting in a reduced cost at keeping wildfires at bay. In the project, a total of 6 machine learning models were created. These models were a simple neural net, a ResNet50 based model, a DenseNet121 based model, a MobileNet based model, and two ensemble models. These models were then transferred to the NVIDIA Jetson Nano for testing and their accuracies, precision, and recall metrics were then compared. Note that the NVIDIA Jetson Nano was chosen because of its support of TensorFlow and Keras, which are machine learning frameworks that the model built during the project use.

The testing models were performed based on two design requirements. These design requirements were an accuracy greater than 90% and an inference time on the NVIDIA Jetson Nano under 1 second. The testing showed that the simple neural net model, the two ensemble models, and the MobileNet model failed to achieve a 90% accuracy. That being said, both ResNet50 and DenseNet121 satisfied both design requirements. Overall, ResNet50 was recommended to be used in the UAV because of its consistency in training and its shorter inference time when compared to DenseNet121.

References

[1] "Cost of wildland fire protection | Natural Resources Canada", *Nrcan.gc.ca*. [Online]. Available: https://www.nrcan.gc.ca/climate-change/impacts-adaptations/climate-change-impacts-forests/forest-change-indicators/cost-fire-protection/17783. [Accessed: 30- Jul- 2020]

[2] C. Yuan, Y. Zhang and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques", Canadian Journal of Forest Research, vol. 45, no. 7, pp. 783-792, 2015.

[3] Y. Zhao, J. Ma, X. Li and J. Zhang, "Saliency Detection and Deep Learning-Based Wildfire Identification in UAV Imagery", Sensors, vol. 18, no. 3, p. 712, 2018.

[4] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho and S. Baik, "Convolutional Neural Networks Based Fire Detection in Surveillance Videos", IEEE Access, vol. 6, pp. 18174-18183, 2018.

[5] Z. Jiao, Y. Zhang, J. Xin, L. Mu, Y. Yi, H. Liu and D. Liu, "A Deep Learning Based Forest Fire Detection Approach Using UAV and YOLOv3", 2019 1st International Conference on Industrial Artificial Intelligence (IAI), 2019.

[6] C. Yuan, Z. Liu and Y. Zhang, "Vision-based forest fire detection in aerial images for firefighting using UAVs", 2016 International Conference on Unmanned Aircraft Systems (ICUAS), 2016.

[7] J. Park, B. Ko, J. Nam and S. Kwak, "Wildfire smoke detection using spatiotemporal bag-of-features of smoke", 2013 IEEE Workshop on Applications of Computer Vision (WACV), 2013.

[8] M. Yin, C. Lang, Z. Li, S. Feng and T. Wang, "Recurrent convolutional network for video-based smoke detection", *Multimedia Tools and Applications*, vol. 78, no. 1, pp. 237-256, 2018.

[9] A. Pundir and B. Raman, "Dual Deep Learning Model for Image Based Smoke Detection", *Fire Technology*, vol. 55, no. 6, pp. 2419-2442, 2019.

[10] Y. Cao, F. Yang, Q. Tang and X. Lu, "An Attention Enhanced Bidirectional LSTM for Early Forest Fire Smoke Recognition", *IEEE Access*, vol. 7, pp. 154732-154742, 2019 [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8865046. [Accessed: 30- Jul- 2020]

[11] T. Toulouse, L. Rossi, A. Campana, T. Celik and M. Akhloufi, "Computer vision for wildfire research: An evolving image dataset for processing and analysis", *Fire Safety Journal*, vol. 92, pp. 188-194, 2017 [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0379711217302114. [Accessed: 30- Jul- 2020]

[12] M. Castelluccio, G. Poggi, C. Sansone and L. Verdoliva, *Land Use Classification in Remote Sensing Images by Convolutional Neural Networks*. 2015 [Online]. Available: https://arxiv.org/pdf/1508.00092.pdf. [Accessed: 30- Jun- 2020]

[13] I. Sevo and A. Avramovic, "Convolutional Neural Network Based Automatic Object Detection on Aerial Images", *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 5, pp. 740-744, 2016.

[14] C. Yuan, Z. Liu and Y. Zhang, "Fire detection using infrared images for UAV-based forest fire surveillance", *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017.

[15] W. Lee, S. Kim, Y. Lee, H. Lee and M. Choi, "Deep neural networks for wild fire detection with unmanned aerial vehicle", *2017 IEEE International Conference on Consumer Electronics (ICCE)*, 2017.

[16] C. Yuan, Z. Liu, A. Hossain and Y. Zhang, "Fire Detection Using Both Infrared and Visual Images With Application to Unmanned Aerial Vehicle Forest Fire Surveillance", *Volume 9: 15th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, 2019.

[17] D. Kinaneva, G. Hristov, J. Raychev and P. Zahariev, "Early Forest Fire Detection Using Drones and Artificial Intelligence", *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019.

[18] P. Chamoso, A. Gonz´alez-Briones, F. De La Prieta and J. Corchado, "Computer vision system for fire detection and report using UAVs", 2018 [Online]. Available: https://www.researchgate.net/publication/326588998_Computer_vision_system_for_fire_detection_and_report_using_UAVs. [Accessed: 30- Jun- 2020]

[19] J. Fernández-Berni, R. Carmona-Galán, J. Martínez-Carmona and Á. Rodríguez-Vázquez, "Early forest fire detection by vision-enabled wireless sensor networks", *International Journal of Wildland Fire*, vol. 21, no. 8, p. 938, 2012 [Online]. Available: https://www.researchgate.net/publication/264977658_Early_forest_fire_detection_by _vision-enabled_wireless_sensor_networks. [Accessed: 30- Jul- 2020]

[20] "NVIDIA Jetson Nano Developer Kit - DFRobot", *Dfrobot.com*. [Online]. Available: https://www.dfrobot.com/product-1909.html. [Accessed: 31- Jul- 2020]

[21] "Bringing the Power of AI to Millions of Devices", *NVIDIA*, 2020. [Online]. Available: https://www.NVIDIA.com/en-us/autonomous-machines/embedded-systems/jetson-nano/. [Accessed: 30- Jul- 2020]

[22] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT Press, 2016, p. 326-366.

[23] "Corsican Fire Database", *Cfdb.univ-corse.fr*. [Online]. Available: http://cfdb.univ-corse.fr/index.php. [Accessed: 30- Jun- 2020]

[24] Denver7, *Raw Video: Garage, car in Black Forest blows up in fire*. 2013 [Online]. Available: https://www.youtube.com/watch?v=up3kuTwBpsw. [Accessed: 30- Jun- 2020]

[25] Need4Life, *Beautiful Drone Flight Over Forest with Pink Floyd Sorrow (solo) 4K UHD*. 2017 [Online]. Available: https://www.youtube.com/watch?v=-vYqyjMmcrY. [Accessed: 30- Jun- 2020]

[26] Sanuck176, *DJI Spark flies over forest fire.*. 2019 [Online]. Available: https://www.youtube.com/watch?v=M97sJdyeEM4. [Accessed: 30- Jun- 2020]

[27] AbimbolaOO and OfafenwaMoses, "DeepQuestAI/Fire-Smoke-Dataset", *GitHub*, 2020. [Online]. Available: https://github.com/DeepQuestAI/Fire-Smoke-Dataset. [Accessed: 30- Jun- 2020]

[28] jivitesh-sharma and mortengoodwin, "cair/Fire-Detection-Image-Dataset", *GitHub*, 2020. [Online]. Available: https://github.com/cair/Fire-Detection-Image-Dataset. [Accessed: 30- Jun- 2020]

[29] A. Saied, "FIRE Dataset", *Kaggle.com*, 2020. [Online]. Available: https://www.kaggle.com/phylake1337/fire-dataset. [Accessed: 30- Jun- 2020]

[30] New York City Files, *Aerial Nature Videos, Amazing Drone Footage of Forests and Meadows*. 2017 [Online]. Available: https://www.youtube.com/watch?v=yZENXvTHkFM. [Accessed: 30- Jun- 2020]

[31] nabesarawak, *Planted Forest l Eagle Eye Project l Borneo l Aerial View*. 2016 [Online]. Available: https://www.youtube.com/watch?v=RW0tLvxf_1E. [Accessed: 30- Jun- 2020]

[32] Flight Media, *Amazing 4k Drone Video of this Canadian Mountain Town | Flights EP. 01*. 2018 [Online]. Available: https://www.youtube.com/watch?v=EMqlDoTOi_M. [Accessed: 30- Jun- 2020]

[33] Just Icelandic, *Drone Footage From Rare Wildfire in Iceland*. 2020 [Online]. Available: https://www.youtube.com/watch?v=9HL7P8ZIKF0. [Accessed: 30- Jun- 2020]

[34] Allison Photo & Aerial Imaging, *Yost Ridge Wildfire*. 2018 [Online]. Available: https://vimeo.com/267869987. [Accessed: 30- Jun- 2020]

[35] CTV News, *Aerial view from CTV News chopper shows raging fire in B.C.*. 2015 [Online]. Available: https://www.youtube.com/watch?v=YyO_iFyraXE. [Accessed: 30- Jun- 2020]

[36] CTV News, *Aerial video shows daunting scale of B.C. wildfire*. [Online]. Available: https://bc.ctvnews.ca/video?clipId=1166908. [Accessed: 30- Jun- 2020]

[37] CTV News, *Aerial footage of massive Alberta wildfire*. [Online]. Available: https://calgary.ctvnews.ca/video?clipId=1692751. [Accessed: 30- Jun- 2020]

[38] "FFmpeg", *Ffmpeg.org*. [Online]. Available: https://ffmpeg.org/. [Accessed: 30- Jul- 2020]

[39] I. Sevo and A. Avramovic, "Convolutional Neural Network Based Automatic Object Detection on Aerial Images", *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 5, pp. 740-744, 2016 [Online]. Available: https://ieeexplore.ieee.org/document/7447728. [Accessed: 30- Jul- 2020]

[40] "Papers with Code - ImageNet Benchmark (Image Classification)", *Paperswithcode.com*, 2020. [Online]. Available: https://paperswithcode.com/sota /image-classification-on-imagenet. [Accessed: 30- Jul- 2020]

[41] C Baldassi, E. M. Malatesta, and R Zecchina, “Properties of the Geometry of Solutions and Capacity of Multilayer Neural Networks with Rectified Linear Unit Activations”, *Phys. Rev. Lett*, vol. 123, no. 17, pp. 1-6, 2019.

[42] C. Ju, A. Bibaut and M. van der Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification", *Journal of Applied Statistics*, vol. 45, no. 15, pp. 2800-2818, 2018 [Online]. Available: https://arxiv.org/abs/1704.01664 [Accessed: 30- Jul- 2020]

[43] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition”, 2016 [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed: 30 - Jul - 2020]

[44] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks", 2016 [Online]. Available: https://arxiv.org/abs/1608.06993. [Accessed: 31- Jul- 2020]

[45] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, 2017 [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: 31-Jul-2020]

Appendix A: Base Model Architectures

ResNet50

Residual networks were created with the purpose of solving the vanishing/exploding gradient problem; as a neural network gets deeper, it becomes increasingly difficult to train through backpropagation due to the gradient becoming too small or large near the front nodes [43]. Instead of just stacking more convolutional layers on top of each other, residual networks use an identity mapping by adding the input of a residual mapping to its output. This increases (rather than changes) the function classes approximated by the network and allows the network to retain what it learned previously through these identity mappings. The ResNet building block is shown in figure 7, and the general ResNet architecture is shown in table 6.

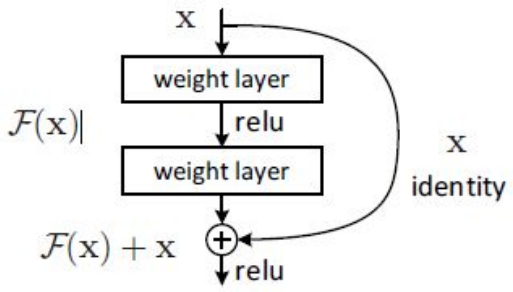

Figure 7: Building block of a residual network [43].

Table 6: Detailed ResNet architecture [43].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

DenseNet121

DenseNet121 is a type of convolutional neural network where each layer is connected to every other layer in a feed-forward fashion [44]. Assuming a normal CNN has L layers with L connections, a DenseNet121 CNN has L layers with L(L+1)/2 direct connections [44]. Note that the 121 in Dense121 denotes the layer depth of the model. For reference, figure 8 provides a visualization of a 5-layer dense block (growth rate of k = 4) and table 7 highlights the specific architecture of DenseNets [44]. The growth rate for the networks is set to k = 32.

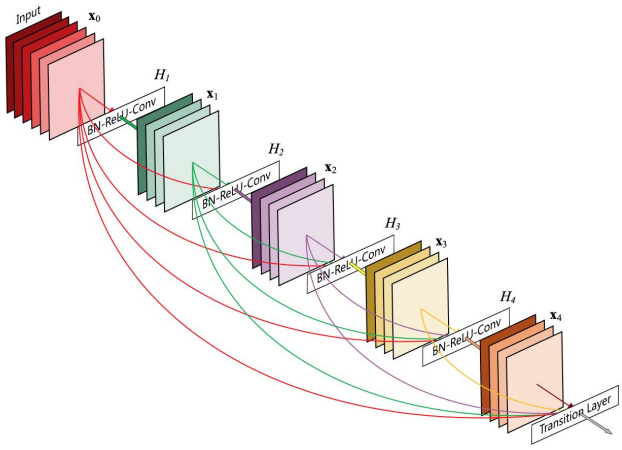

Figure 8. Visualization of a 5 layer dense block [44] .

Table 7. Detailed DenseNet121 architecture [44].

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

MobileNet

A MobileNet is a type of convolutional neural network with a lightweight architecture that was built with efficiency in mind for applications in real time systems [45]. MobileNet's depth wise separable convolutions enable it to have low inference times and be small size relative to other architectures [45]. Depthwise separable convolutions are a "form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1x1 convolution called a pointwise convolution" [45]. For reference, figure 9 provides a visualization of the pointwise convolution building blocks of MobileNets and table 8 highlights the specific architecture of MobileNet .

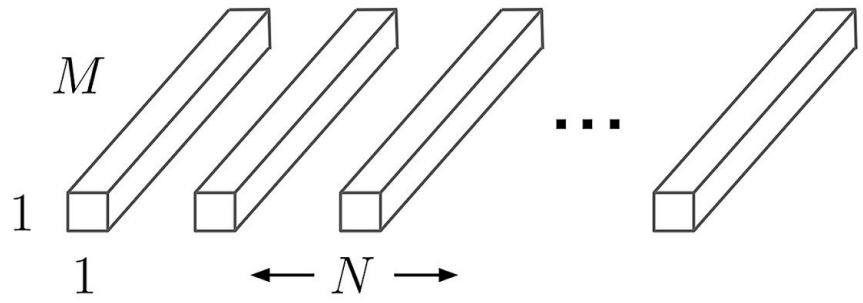

Figure 9. Pointwise convolution building blocks of MobileNets [45] .

Table 8. Detailed MobileNet architecture [45].

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Appendix B: Test Accuracy and Hidden Layers

Initial model prototypes contained no hidden layers. Consequently, this resulted in little more than a logistic regression classifier being built on top of a feature extractor. Using this configuration, ResNet50, DenseNet121, and MobileNet achieved respective accuracies of 0.9323, 0.9198, and 0.8772. The tables 9 to 11 show the resulting test accuracies of each model when compiled with 12 different hidden layer configurations. Of particular interest is the MobileNet model, whose highest accuracy was observed with no hidden layers. In fact, as more layers were added onto the classifier, the accuracy quickly decreased and became wildly inconsistent with subsequent training sessions. This may suggest that using MobileNet as a feature extractor allows the data to be linearly separable, while the other two base models require nonlinearity.

Table 9: Model accuracies by hidden layer configuration: ResNet50.

	30 Nodes	60 Nodes	90 Nodes	180 Nodes
1 Layer	0.8872	0.9348	0.9248	0.9298
2 Layers	0.9123	0.8847	0.9223	0.9273
3 Layers	0.9248	0.9098	0.9023	0.8872

Table 10: Model accuracies by hidden layer configuration: DenseNet121.

	30 Nodes	60 Nodes	90 Nodes	180 Nodes
1 Layer	0.9198	0.9424	0.9273	0.8897
2 Layers	0.8822	0.8672	0.9398	0.9123
3 Layers	0.9073	0.9323	0.8972	0.8722

Table 11: Model accuracies by hidden layer configuration: MobileNet.

	30 Nodes	60 Nodes	90 Nodes	180 Nodes
1 Layer	0.7969	0.8296	0.8195	0.7469
2 Layers	0.7393	0.8296	0.7168	0.8197
3 Layers	0.7444	0.7469	0.7845	0.7719

Appendix C: Confusion Matrices

Appendix C contains the confusion matrices that were used to calculate the precision and recall in Table Q. The confusion matrices are displayed in tables 12-17. Note the terminology used in the tables are TP (true positives), FP (false positives), TN (true negatives), and FN (false negatives).

Table 12. ResNet50 Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	179 (**TP**)	21 (FN)
No Fire (actual)	4 (FP)	195 (**TN**)

Table 13. DenseNet121 Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	182 (**TP**)	18 (FN)
No Fire (actual)	6 (FP)	193 (**TN**)

Table 14. MobileNet Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	175 (**TP**)	25 (FN)
No Fire (actual)	27 (FP)	172 (**TN**)

Table 15. Simple Neural Net Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	146 (**TP**)	54 (FN)
No Fire (actual)	84 (FP)	115 (**TN**)

Table 16. Ensemble Voting Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	186 (**TP**)	14 (FN)
No Fire (actual)	6 (FP)	193 (**TN**)

Table 17. Ensemble Pooling Confusion Matrix.

	Fire (predicted)	No Fire (predicted)
Fire (actual)	187 (**TP**)	13 (FN)
No Fire (actual)	6 (FP)	193 (**TN**)