# Real-time Fire Detection and Localization within Video Imagery via Deep Convolutional Neural Networks

*Abstract –*

**Context/Background** - The increased use of widespread video monitoring in urban areas and industrial sites leads to the possibility that such systems could be used for the automatic detection of fire as a secondary detection system in addition to traditional fire alarm systems, providing numerous advantages to a solely traditional detection approach. Existing solutions to this problem do not perform strongly enough to be considered reliable and so there is room for improvement.

**Aims** - The aim of this project is to extend the current state of the art in visual fire detection by investigating the use of deep convolutional neural networks for this task. We aim to assess to what extent current state of the art deep convolutional neural network techniques can be adapted to perform the overall end-to-end classification task of detecting and localizing fire within video imagery.

**Method** - Established test imagery and data gathered from online sources is labelled frame-by-frame and used to train various deep convolutional neural network architectures to detect and localize the presence of fire within a given image. In addition to this, new model architectures are constructed with the aim of increasing solution efficiency, and their performance evaluated in comparison to existing models.

**Results** - An improvement upon the current state of the art solution to visual fire detection (Chenebert et al. 2011) is produced by using deep convolutional neural networks, with a maximum accuracy of 93.4% for binary detection, and 88.6% for localization. Additionally, two new deep convolutional neural network architectures are produced by simplifying existing architectures, which increase classification speed by factors of 4.2 and 3.3 when applied to this problem whilst achieving similar performance.

**Conclusions** – The solution improves upon the current state of the art in visual fire detection by applying deep convolutional neural networks to the problem. Simplifications to complex state of the art deep convolutional neural network architectures are made to improve classification speed whilst preserving performance in binary problems such as this.

*Keywords* – fire detection, fire localization, deep convolutional neural networks, computer vision, deep learning, machine learning, object recognition.

## I. Introduction

Fires, whether natural or man-made, still pose a significant danger to society, and have the potential to cause large-scale damage. The occurrence of fires cannot ever be completely prevented, so we resort to relying on systems that execute some form of alert when a fire has ignited, and then an effort is made to extinguish the fire.

Traditionally, modern fire detection is performed using a smoke, or heat, detector. Smoke detectors rely upon smoke from a fire blocking the ionization between two detector plates, causing a loss of current; whereas heat detectors rely upon the heat given out by a fire heating up a sensitive element within the detector. These methods of fire detection rely entirely upon transmission through air (smoke or heat), and so require that the detector is placed within close proximity of the fire in order to detect its presence. Consequently, it is impossible to detect fires in outdoor spaces, and it is possible that a fire within an indoor area is only detected at a certain time after the fire has ignited – giving it time to spread before an alarm is raised.

Using video imagery as a means of fire detection, rather than traditional methods, poses several advantages. Firstly, a visual representation of a fire provides details of the nature of the fire which may be useful in determining what kind of response is required. This could lead to more efficient use of time, and may reduce the damage caused by a fire, due to there being a more appropriate response. Additionally, because visual detection does not require the use of any physical medium for transmission (other than light, for which transmission time is negligible), there need not be any delay between fire ignition and detection. This leads to a much faster response time than traditional methods, which reduces the risk of additional damage within the period between ignition and detection. Another advantage of being independent of a physical medium is that visual detection can be used to detect fires in large outdoor areas, where traditional methods would be completely ineffective. This is of particular use in the monitoring of large regions of woodland to detect forest fires, or in monitoring outdoor industrial compounds.

Visual fire detection can be implemented using CCTV, which is already widely in use, and is prevalent in many public and private areas for surveillance purposes. Consequently, additional cameras for gathering video data would not need to be implemented, meaning that this solution could be applied with insignificant infrastructure expenses. Another possible application of visual fire detection is within autonomous vehicles/agents. These agents often interact with their environment by utilizing visual information, and so it would be extremely easy to integrate a visual fire detection system. This could eventually lead to the possibility of producing firefighting agents/drones in order to combat fires without human risk.

There has been significant prior research on possible strategies to realise the implementation of a visual fire detection system, however no strategy yet exists which is reliable enough to be a candidate for widespread implementation. This work aims to produce a solution to visual fire detection which is reliable enough to be considered a candidate for fire detection in the real world, either in conjunction with traditional methods or exclusively.

### *A. Background*

The motivation to apply deep neural networks (DNN) to this problem stems from the recent surge in popularity, and effectiveness, of deep convolutional neural networks (DCNN) as a computer vision technique. DCNN are currently the state of the art in computer vision – massively outperforming all other methods on tasks such as object detection and localization. Since the work of AlexNet (Krizhevsky et al. 2012), all of the winners of the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) (Russakovsky et al. 2015) have

utilized DNN. The ILSVRC is a renowned computer vision competition in which researchers and organizations from all over the world showcase their computer vision techniques, and compete to find the best solution for tasks such as object recognition and localization. It has become the standard for evaluating the strength of computer vision techniques, which are trained on a subset of the ImageNet (Deng et al. 2009) database – a 20,000 class dataset comprised of over 14,000,000 images. Over the past two years, the winners of this challenge have surpassed expert human performance in various computer vision tasks, and so these techniques are likely to perform excellently when applied to visual fire detection.

DNN are an extension of the traditional artificial neural network, in that they contain a number of hidden layers between the input and output layers. The number of hidden layers varies depending on the implementation, however, the recent trend has seen the depth of networks increase significantly – with some networks being up to 152 layers deep (He et al. 2015). Previous work has shown that artificial neural networks produce an effective solution when applied to the problem of visual fire detection (Zhang et al. 2009; Chenebert et al. 2011), so it follows that DNN should also perform well in this task. Additionally, unlike methods used in previous approaches, DNN perform an end-to-end classification of images, so there is no need to manually extract candidate fire regions before feeding data into the classifier.

### *B. Objectives*

The research question proposed is: *Can deep neural networks be used for the task of detecting and localizing fire within video imagery in order to advance the current state of the art?*. To address this research question, the objectives for this project were divided into three categories: minimum, intermediate, and advanced.

The minimum objective of this project was to establish a candidate DCNN architecture from the literature which would likely perform well for this task, and implement this model. This implemented model should then be trained using data from previous work (Chenebert et al. 2011), and data from an established visual fire detection evaluation dataset (Steffens et al. 2015). This model was required to produce a binary classification of any given image, i.e. determine if the image was to be classified as *fire* or *no fire*, and was expected to achieve an accuracy of at least 70% when evaluated. The purpose of this objective was to establish a strong understanding of DNN architectures, and build a foundational model for the workflow of the project.

The intermediate objectives were to research and implement other DCNN architectures from the literature, evaluate their performance in visual fire detection, and compare them. Also, the construction of a new training dataset should be undertaken in order to provide a more extensive range of training data with the expectation of improving performance. In addition to the construction of this new dataset, a validation dataset should be constructed, which is independent of any training data, in order to accurately compare different implementations. One or more of the trained models should achieve an accuracy that at least matches the current state of the art of 87.3% (Chenebert et al. 2011).

The advanced objectives were to extend the current approach by performing localization of fire within images, rather than just a binary classification. Also, the efficiency of approaches should be evaluated, and variations should be made to existing architectures with the aim of increasing efficiency, whilst preserving performance. The motivation behind this objective is that the computational power of cameras, which will be performing the detection, will be extremely low, and so a less computationally expensive classification is preferable. It was also expected that at least one of the trained models improves upon the current state of the art in fire localization within video imagery (Chenebert et al. 2011).

## II. Related Work

There has been significant academic effort into producing a visual fire detection system which is a candidate for implementation in the real world, and previous solutions to this problem have had varying degrees of success. To classify an image as either *fire* or *not fire* the image must be analysed, and features extracted. Generally, previous solutions use either colour features or temporal features in order to perform a classification, some approaches utilize both, but place more emphasis on one than the other.

### *A. Colour-based Approaches*

The most basic approach for identifying candidate fire pixels within an image is by using colour analysis. An early example of this technique presented a set of decision rules in order to identify fire regions within an image (Chen et al. 2003) using the RGB (red/green/blue) and HSI (hue/saturation/intensity) colour spaces. These rules were simple and easily computable, such as $R \geq G > B$, and a pixel was determined to be a candidate fire pixel if all rules were satisfied. This idea was expanded to involve a unimodal Gaussian distribution (Ko et al. 2009) and alternative colour spaces were investigated, such as the YCbCr (luma/blue-difference/red-difference) space, in which similar rules were introduced (Çelik & Demirel 2009). More recent approaches use colour-based rules to extract candidate fire pixels as part of a more complex solution (Morerio et al. 2012; Chenebert et al. 2011). Traditionally, image classification was performed by applying a set of conditions to the candidate fire pixels; whereas modern approaches perform classification by using the candidate fire pixels as input to machine learning classifiers.

A reliance on colour-based image analysis to extract candidate fire pixels imposes solution limitations, due to the fact that this method generates a large percentage of false positive results. False positive rates such as 31.5% (Çelik & Demirel 2009) are enabled by the misclassification of objects which have a similar colour to fire.

### *B. Temporal-based Approaches*

Approaches which use mainly temporal information to identify candidate fire pixels exploit the fact that fire has a flickering motion, and so consecutive frames from a video of a fire will vary significantly. Basic approaches to utilizing temporal information for this purpose include simple image differencing (Foo 1996), and measuring *fire growth* by comparing the number of fire pixels between frames (Phillips et al. 2002) – both of which are used in conjunction with colour-based methods. Later approaches made use of more sophisticated techniques such as Markov models (Toreyin et al. 2005), and representing fire boundaries using Fourier coefficients to calculate the stochastic characteristics of the boundary motion (Ahuja 2004). Typically, these approaches apply a set of conditions involving the candidate fire pixels in order to classify the entire image or image sequence.

These methods produce relatively successful solutions, but are usually implemented as part of a more complex solution which combines multiple approaches. There are two major limitations of temporal-based solutions to this problem. Firstly, these methods require that multiple frames are analysed, and so classification cannot be performed upon a single image. Secondly, they require that the video footage is captured by a camera which is stationary, and that the background of the video is stationary – this makes these methods inapplicable to autonomous agents.

### *C. Application of Machine Learning Classifiers*

The classification of an image as *fire* or *not fire* is performed by utilizing feature information which is extracted from the image. Early approaches to classification involved simply applying a set of conditions to extracted features, such as *IF rule1 AND rule2 THEN x ELSE y* statements (Chen et al. 2003). These approaches have the downside of being dependent on the creator of the conditions being completely correct, and that it is unlikely that a set of conditions can cover every possible set of features. More recent approaches, however, have used image features as input to machine learning classifiers – leading to more successful solutions.

Successful applications of machine learning classifiers to this problem have employed simple neural networks with a single hidden layer. Zhang *et al*. used one such network to classify a feature vector created by analysing the area, roundness, and contours of fire regions (Zhang et al. 2009). A similar classifier was also used by Morerio et al, but they chose to input a feature vector composed of colour and motion information (Morerio et al. 2012). Support vector machines have also been used as a classifier, with inputs such as a vector constructed from temporal variation features with wavelet coefficients (Ko et al. 2009). Chenebert *et al*. proved that decision tree classifiers were extremely effective when applied to this problem by presenting a solution which achieved a mean detection rate of 87.3% - which is the current state of the art solution (Chenebert et al. 2011). The input to this classifier consisted of a combination of colour features extracted by using thresholding and histograms, and texture features extracted in the form of a Grey Level Co-occurrence Matrix (Haralick et al. 1973).

Overall, the application of machine learning to the problem of visual fire detection has led to a significant improvement in the performance of proposed solutions. This is because these classifiers automatically learn a set of rules which are used to classify each image. However, using these classifiers introduces a whole new set of issues such as the possibility of overtraining, the need for more data, and the increased solution complexity which can lead to a classification taking much longer.

### *D. Deep Convolutional Neural Networks*

DCNN were originally established as a computer vision technique when they were used to classify 28 x 28 x1 pixel images of handwritten digits (LeCun et al. 1998). Since Graphical Processing Unit (GPU) programming was not available at this point, the size of inputs to networks had to be restricted greatly in order to make the computation of deep architectures realistically possible. However, more recently, Krizhevsky *et al*. introduced AlexNet (Krizhevsky et al. 2012), which was an 8-layer deep architecture that utilized GPU programming to perform relatively fast computation of network parameters. AlexNet was the first artificial neural network that performed exceptionally well on the ImageNet dataset; producing a top-5 test error rate of 15.3% in the 2012 ILSVRC. Following the success of AlexNet, architectures have become progressively deeper and more effective, with GoogLeNet (Szegedy et al. 2015), an architecture with a depth of approximately 100 layers, achieving a top-5 error of 6.67% in the ILSVRC 2014. The latest DCNN models such as the 152-layer deep architecture introduced by He *et al*. have been shown to outperform human expert performance (He et al. 2015). The model of He *et al*. achieved a top-5 error of 3.6% in the 2015 ILSVRC, where human performance lies within the 5-10% range.

Every previous solution to visual fire detection has relied upon the manual extraction of fire features, and using these features as input to a classification method. These methods have the inherent drawback that the features being manually extracted may not necessarily be the

optimal features to identify fire. Our solution does not require that features are extracted manually as DNN perform end-to-end image classification. The success of other machine learning techniques which have been applied to this problem suggests that DCNN will perform just as well, if not better, than these solutions. This is due to the fact that current state of the art DCNN (Szegedy et al. 2015) are capable of out-performing any other machine learning technique in computer vision tasks such as object recognition. We present a solution which is independent of any previous solution to this problem by applying DCNN.

## III. SOLUTION

In order to produce a solution to the problem of fire localization with video imagery, a specific workflow would need to be implemented. Firstly, an image would be segmented to produce candidate fire regions. Secondly, a DNN would classify these segments to determine whether they contained fire. Finally, these segment classifications could be used to localize fire within the image. However, rather than beginning to solve the first step and producing a method of segmenting images to use as input to a DNN model, we began with the second step and implemented a DNN which performs binary classifications. This enabled the production of an intermediate solution to our problem. This solution is in some ways superior to a localization solution, as fire detection can be performed without the added complexity of localizing the fire, and so can be performed faster. The downside of this solution is that it is not applicable to autonomous agents, which would require the exact location of a fire to act.

### *A. Implementation Tools*

There are several open source machine learning libraries which were available for use in our implementation, however we decided to use TensorFlow (Tensorflow.org, 2017). This decision was made because TensorFlow is currently by far the most supported machine learning library, and is designed with the intention of making neural networks easy to implement. More specifically, TF Learn, a high-level library for TensorFlow which enables the simple implementation of very complex model architectures was used. This library allowed us to implement existing architectures and easily make modifications to them, without the additional complexity of programming in low-level TensorFlow. We used OpenCV (Opencv.org, 2017), a popular open source computer vision library, for all image and video manipulation. Consequently, the Python language was used for all implementation, as both TensorFlow and OpenCV have Python interfaces, and the TensorFlow documentation was limited for other languages.

### *B. Deep Convolutional Neural Networks*

A thorough explanation of how DCNN function is beyond the scope of this work, however we will provide a brief summary in order to offer a better understanding of the concept. DNN are an extension of the traditional artificial neural network, which is a technique for pattern recognition which attempts to replicate the function of neurons in the human brain. A network is composed of many elements called perceptrons, each of which simulates the activity of a neuron in the human brain. Each perceptron takes an input, and produces an output by applying its *activation function* to the input, which is then typically sent as input to another perceptron via a weighted edge. Typically, perceptrons are organized into layers, with a *feed-forward* architecture, meaning that the output of a perceptron in a given layer is only sent to the next layer in the network. Each perceptron usually has connections to every perceptron in the previous layer. Consequently, in a given network there

would typically be an input layer, an output layer, and a number of *hidden* layers between the input and output layers. These networks learn by computing the output for a given input, and then updating the connection weights depending on the difference between the actual output and the expected output, in order to better meet the expected output if a similar input is received (LeCun et al. 1990). We call this *supervised learning* - feeding data into the network along with labels which contain the expected output. With each training iteration, the network becomes better at classifying that specific input by adjusting connection weights – the network *learns* how it should classify each input.

A *deep* neural network only differs from a standard neural network in that there are a greater number of hidden layers between the input and output layer of the network. Since each layer of perceptrons trains on the set of features which are based on the output of the previous layer, the deeper the network (i.e. the more layers it has), the more complex the features it can recognize. Therefore, when we process complex data such as images, DNN tend to perform better than standard neural networks.

This is how DNN work with a general input, however to process complex images we must make use of a more sophisticated structure. Due to images having three dimensions (width, height, and depth), they contain a huge number of parameters, and so processing an image with a standard network would be very inefficient. To enable neural networks to efficiently process images, we use something called *convolutional* neural networks (LeCun et al. 1998), in which the responses of each perceptron are produced by performing a convolution operation upon the image. This allows us to arrange perceptrons in three dimensions (width, height, and depth), and therefore massively reduce the number of parameters required to process an image.

There are typically 3 main types of layers that are used to build DCNN, which will be used throughout our solution. *Convolutional layers* perform convolution by sliding a filter across the image and computing the dot product of the filter with the image values at each point, this produces a two-dimensional activation map which gives the response of the filter at each point in the image. Many of these filters can be present in a single layer, each producing a separate activation map. As the network is trained, it will learn filters that activate when they see specific features within the image, such as circular patterns or vertical edges. Convolutional layers are the backbone of the convolutional neural network. *Pooling layers* reduce the dimensions of their input by separating the input into grids, and performing a *down-sampling* operation upon each grid. This operation is usually the *max* function, which returns the maximum of a set of numbers. These layers reduce the number of parameters in the network, and help prevent overfitting. *Fully-connected layers* typically occur after the convolutional and pooling layers, and have full connections to all activations in the previous layer, similar to a layer in a basic neural network. They allow us to shape the output of the network to fit the number of classes in the problem, which in our case is 2.

All networks presented in our solution are DCNN with varying structures.

### *C. Database Compilation*

Central to the success of DNN, and all other machine learning methods, is a high-quality labelled dataset.

When compiling a dataset to perform training, we first searched for readily available datasets within the literature which have already been proven to be effective. This led us to include the dataset used to construct the current state of the art solution (Chenebert et al. 2011), which consisted of four videos –producing a total of 75,683 images. We also included an established visual fire detection evaluation dataset (Steffens et al. 2015), which was constructed solely to evaluate the effectiveness of different solutions to this problem. The

crucial advantage of this dataset was that it would allow us to compare directly to previous solutions, including the current state of the art solution. Also, this data was already entirely labelled with bounding boxes, which meant that less time had to be spent labelling images.

We wanted to increase the variation in image features to produce a solution which was as robust as possible, so YouTube (YouTube.com 2017) was used to source additional data, and a dataset consisting of 21 videos was constructed – producing a total of 269,426 images. Videos of different environments and different types of fire such as forest fire, house fire, helmet camera fire etc., as well as videos which did not contain fire, were selected. This gave enough data for all training procedures and ensured that, after this preliminary work, the issue of compiling data would not have to be revisited.

To complete the dataset and enable training, all images had to be labelled. A label of 1 was assigned if an image contained fire, and a label of 0 was used otherwise. At this stage, labels were encoded by including a text file in the video directory, with the same title as the video, which had on each line the frame number and its corresponding label. To produce labels for the dataset created by Steffens *et al*., a Python program was written which simply encoded a 1 if bounding box data existed for a given frame, and 0 otherwise. To label the remainder of the data, frames from each video were extracted into separate directories using OpenCV, which made it easy to scroll through each directory and see which frames contained fire by glancing over the thumbnails. Finally, the frame numbers in which fire was present in each video were added to a Python program, which produced a label file for each video.

Our dataset comprised of a total of 365,702 images, which were of various dimensions. Since any given DCNN must have a fixed input dimension, all images were required to be of identical dimensions. Each image was resized to a dimension of 224 x 224 x 3, as these were the dimensions used by the highest performing DCNN architectures (Simonyan & Zisserman 2015; Szegedy et al. 2015; Krizhevsky et al. 2012). We experimented with larger image dimensions but this did not lead to an improvement in results, so we decided to continue using these small images to reduce the complexity of our solution. Additionally, we faced the problem that the frames from a single video would all be very similar, and therefore most of the frames would be superfluous. To compensate for this, we extracted at most 500 frames from each video at constant intervals in order to provide a sample of the overall content without having duplicate data. After this elimination, the final training dataset consisted of 23,408 images of equal dimensions. To enable the transfer and efficient use of the dataset, the data was encoded in Hierarchical Data Format, a scientific data format used to store and organize large amounts of data, which significantly reduced the space occupied by the data file.

In order to evaluate various models upon a common medium, a validation dataset was constructed which comprised of 6 videos from YouTube, and featured a variation of fire and environment types. Frames were extracted in a similar way to those of the training dataset, producing a total of 2931 images. This set was completely independent of the training set, so that the effectiveness of our solution could be evaluated fairly.

### *D. Network Architectures*

When searching for candidate architectures, we looked mainly at performance in the ILSVRC, but also selected architectures which had varying design principles in order to see which type of design best fitted our problem, and that could be implemented in TensorFlow.

**AlexNet** (Krizhevsky et al. 2012) was chosen as a starting point due to its simplicity and the fact that many later architectures reused principles which were introduced here. In the original AlexNet architecture, the hidden layers were split into two branches to enable the model to be trained using two GPU's. We implemented the model with a single branch, due to hardware restrictions. The original architecture is illustrated in Figure 1. Initially, a convolutional layer with a kernel size of 11 is followed by another convolutional layer of kernel size 5. The output of each of these layers is followed by a max pooling layer and *local response normalization*. Three more convolutional layers then follow, each having a kernel size of 3, and the third is followed by a max pooling layer and local response normalization. Finally, three fully connected layers are stacked to produce the output. This network is relatively simple – containing only five convolutional layers and three fully connected layers.

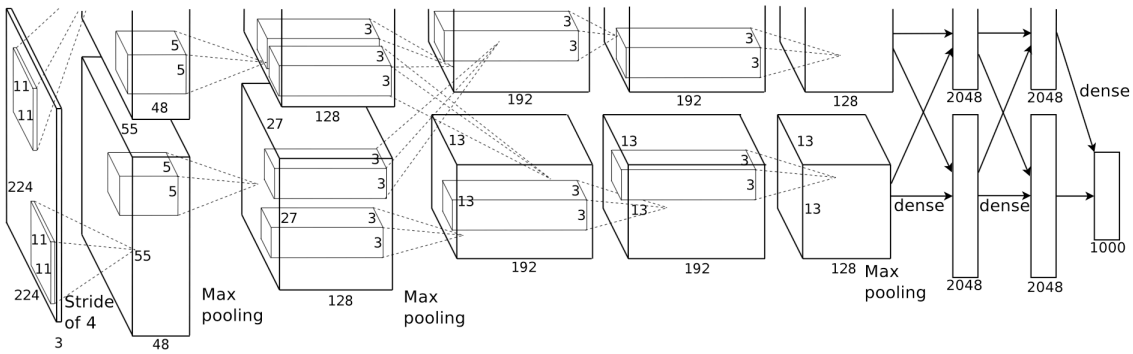


**Figure 1.** An illustration of the original AlexNet architecture (Krizhevsky et al. 2012).

**VGG (Visual Geometry Group) Net** (Simonyan & Zisserman 2015) was based on the principle of prioritizing simplicity and depth over complexity – all convolutional layers had a kernel size of 3, and the network had a depth of 16 layers. This model consists of groups of convolutional layers, and each group is followed by a max pooling layer. The first group consists of two convolutional layers, each with 64 filters, and is followed by a group of two convolutional layers with 128 filters each. Subsequently, a group of three layers with 256 filters each, and another two groups of three layers with 512 filters each feed into three fully connected layers which produce the output. The architecture of VGG Net is illustrated in Figure 2. We decided to implement the 13-layer version of this network, due to the 16-layer model containing so many parameters. This simply involved removing one layer from each of the final three groups of convolutional layers.

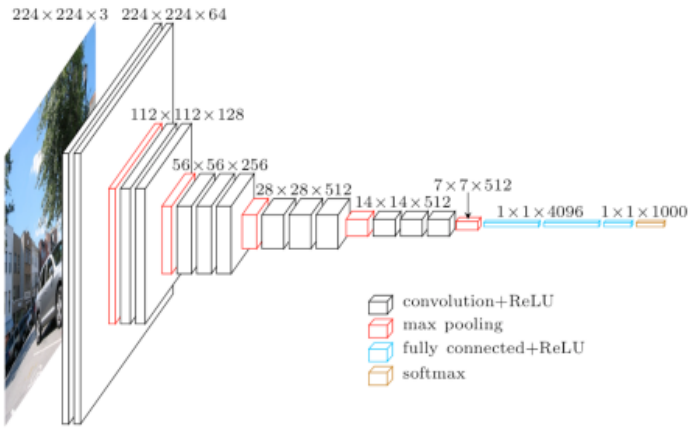


**Figure 2.** An illustration of the VGG Net architecture.

**GoogLeNet** (Szegedy et al. 2015) was implemented because of its impressive performance in the ILSVRC, as well as the fact that it uses the principle of executing elements of the network in parallel, rather than sequentially. The GoogLeNet model is composed almost entirely of a single repeating element, called an *Inception module*. This module consists of four parallel strands of computation, each containing different layers. The theory behind this choice is that rather than having to choose between convolutional filter parameters at each stage in the network, multiple different filters can be applied in parallel and their outputs concatenated.

Different sized filters may be better at classifying certain inputs, so by applying many filters in parallel the network will be more robust. The architecture of an Inception module is illustrated by Figure 3. The four strands of computation are composed of convolutions of kernel sizes 1 x 1, 3 x 3, and 5 x 5, as well as a 3 x 3 max pooling layer. 1 x 1 convolutions are included in each strand to provide a dimension reduction – ensuring that the number of outputs does not increase from stage to stage, which would drastically decrease training speed. 9 of these inception moduels are placed in sequence in the overall model architecture, a full explaination of which is provided by Szegedy *et al*.

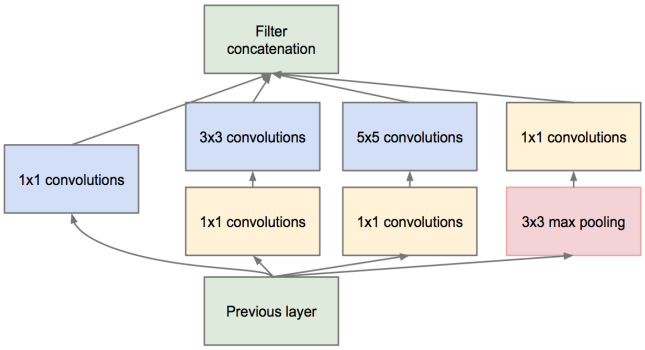


**Figure 3.** An Inception module (Szegedy et al. 2015).

Implementing the above 3 architectures gave us a broad spectrum of DCNN designs with which to tackle our problem. AlexNet is a relatively shallow network with large convolutional filters, VGG Net is deep and simple – only having convolutional filters with 3 x 3 kernel size, and GoogLeNet is comparatively complex, applying the principle of executing elements of the network in parallel. These architectures were directly applied to the binary classification problem of detecting whether fire is present in an image. To enable this, the final layer of each model was modified to only output a 2-class prediction, rather than a 1000-class prediction.

### *E. Improving Performance*

Reducing the computational complexity required to produce a classification is of great importance because our solution would be implemented within a camera or on-site without specialised hardware. This can be done by reducing the number of parameters which are used within a given DCNN architecture, however there is usually a trade-off between complexity and performance of the model. A TensorFlow implementation of the VGG-13 model, for example, contains 128,959,042 parameters. Therefore, this model will take much longer to perform a classification than a more lightweight model such as GoogLeNet, which contains only 6,073,906 parameters – a reduction in parameters by a factor of around 21.

With the aim of reducing model complexity whilst preserving performance in our solution, we constructed two new DCNN architectures by adapting principles from, and simplifying, existing architectures.

**Simplified AlexNet (BinNet)**

To design this model, we removed different combinations of layers from AlexNet and investigated the impact upon performance and speed. This simple architecture contains only three convolutional layers, with filters of kernel sizes 5 x 5, 4 x 4, and 1 x 1 respectively, and with number of filters 64, 128, and 256. Each convolutional layer is followed by a max pooling layer with a kernel size of 3 and local response normalization. The convolutional layers are followed by two fully connected layers, each with 4096 incoming connections and tanh activation. A dropout of 0.5 is applied across these two fully connected layers in order to reduce overfitting. Finally, a fully connected layer with 2 incoming connections and softmax activation produces the output. The architecture of BinNet is illustrated in Figure 4.

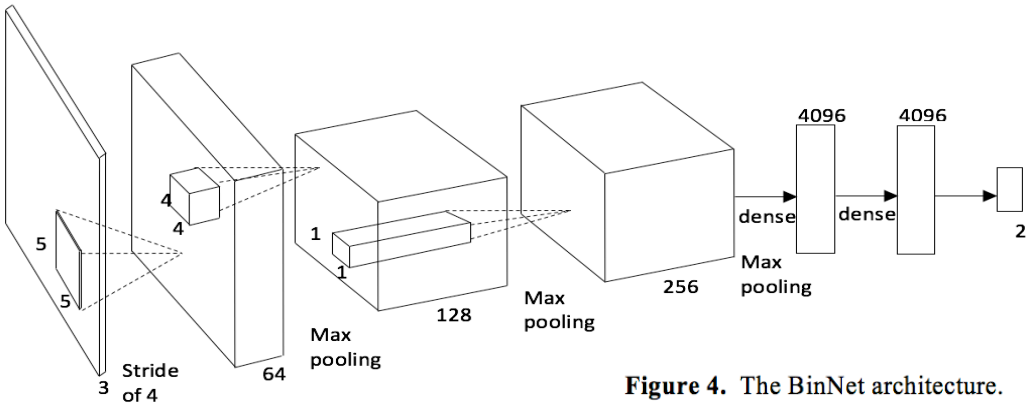


**Figure 4.** The BinNet architecture.

**Simplified GoogLeNet (GoogLeNet-2)**

We simplified the GoogLeNet architecture in order to reduce the number of parameters without greatly altering performance. After experimenting with models featuring different numbers of Inception modules we found that using 3 inception modules, rather than 9 as in the standard model, was optimal. This alteration only reduced performance slightly, whilst greatly reducing the number of parameters.

### *F. Performing Localization*

Extending our binary classification solution to perform localization required that we find a means of splitting a given image into regions, which can then be fed into a binary classifier, and classifications can be made per region. One possible solution to this problem would be to use the traditional method of splitting an image into a grid, and simply classify each individual cell of this grid. However, a pixel grid is an arbitrary segmentation and is therefore not a natural representation of image content. Additionally, using a grid method would only enable us to define a bounding box which encloses the fire, rather than an exact outline. This is a major issue in our problem, as fire does not tend to follow a structure which can be bounded easily by a box – it could be present in many shapes throughout the image.

*Superpixel* (Achanta et al. 2012) based techniques segment an image into a collection of superpixels, which are perceptually meaningful regions of the image. Each superpixel is composed of a group of pixels which are similar in colour and texture, and so are likely to belong to the same object – allowing us to segment an image into regions which each belong to a specific object. Figure 5 (a) and (b) illustrate the segmentation of an image containing fire by using superpixel segmentation.

(a) (b) (c)

**Figure 5.** An image containing fire (a), a superpixel segmentation of the image (b), and an example of the isolation of an individual superpixel (c) which can be used to train a classifier.

We performed superpixel segmentation by using scikit-image (Scikit-image.org 2017), an image processing library for Python. Various algorithms for performing superpixel segmentation exist, however we chose to implement *simple linear iterative clustering* (SLIC) (Achanta et al. 2012) as it produces similar results to that of other methods, yet is faster and more memory efficient. An explanation of this algorithm is beyond the scope of this work, but it is essentially an adaptation of the *k-means* method, with reduced distance calculations due to limiting the search space to a region proportional to the superpixel size.

To adapt our binary classification solution to recognize whether a given superpixel contains fire, we utilized the dataset introduced by Steffens *et al*. which had bounding-box labels. We extracted superpixels from each image in the bounding-box labelled training dataset, and each superpixel was placed over a black background so that the output image dimensions were equal to the original image dimensions, as in Figure 5 (c). To determine whether a given superpixel contained fire we used the bounding-box labels – if a superpixel was entirely contained within the bounding-box, then the superpixel contained fire. This enabled the production of labels for each superpixel image. GoogLeNet-2 was trained upon this new dataset of superpixels in an identical manner to how training was performed in our binary classification solution.

This training method produced a binary classifier which was capable of determining whether a given superpixel contained fire. Therefore, localization could be performed by first segmenting an image into superpixels, then feeding each superpixel into the classifier to determine whether it contained fire. To evaluate our localization solution, an evaluation set was extracted from the Steffens *et al*. dataset, and each image in this dataset was segmented into superpixels which were fed into the classifier. Then, a bounding-box was constructed using the superpixels which were identified as fire, and compared to the bounding-box label to determine the effectiveness of the localization.

### *G. Network Training*

Due to the computational strain of training DNN, the hardware capability we required exceeded that of any standard computer. GPU's are usually used for this purpose due to their ability to execute many operations in parallel. In order to train our models, we made use of a single NVIDIA Titan XP GPU, which provided a huge increase in training speed.

When training a neural network, care must be taken regarding how long the network is trained for with a given dataset. If the network is not trained for long enough then it will have insufficiently learned the patterns in the data needed to effectively perform classifications. Whereas, if the network is trained too much then *overtraining* will occur. Overtraining is the process of training a classifier for so long on a given dataset that it learns a set of rules to *exactly* classify each example in the training data, and so loses all generality. To find the point between which each of our models was insufficiently trained and was overtrained, we trained each model for many *epochs* (a single pass of the entire training set) whilst evaluating performance at regular intervals. This allowed us to find the point at which each model performed optimally.

### *H. Network Testing/Evaluation*

When an image is fed into a convolutional neural network, the output produced is usually in the form of a *prediction vector*. This vector will have a length which is equal to the number of classes in the problem that the network is trying to solve - our problem has 2 classes, so our prediction vector has a length of 2. Each entry of the prediction vector represents a confidence score for the corresponding class. For example, if our network decides that there

is a 90% chance that an input image does not contain fire, and a 10% chance that it does, then it will produce an output vector of [0.1, 0.9]. The sum of all entries in an output vector is always equal to 1.

Evaluation of each model was performed using our evaluation dataset, which was completely independent of any data in the training set. To perform evaluation of a given model, the model checkpoint and the evaluation data were loaded into a Python file. Subsequently, each image in the evaluation set was fed into the model and the entries of the resulting prediction vector were rounded to the nearest integer and compared to the corresponding image label. This allowed us to record the number of true positive, true negative, false positive, and false negative classifications each model made, which could be used to produce various statistical measures of performance.

## IV. Results

To enable the evaluation of our solution, we will produce accuracy (Acc), precision (Prec), true positive rate (TPR), and false positive rate (FPR) statistics which can be calculated using the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) classifications of our models. Also, to provide a more in-depth analysis, the *F-Measure* (F) (Eq. 1) and *Matthews Correlation Coefficient* (MCC) (Eq. 3) (Steffens et al. 2015) statistics will be produced, along with the *Kappa* statistic (Eq. 2) (Bland 2008).

$$F_1 = \frac{2 \times PPV \times TPR}{PPV + TPR} \qquad (1)$$

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \qquad (2)$$

$$MCC = \frac{T_P T_N - F_P F_N}{\sqrt{(T_P + F_P)(T_P + F_N)(T_N + F_P)(T_N + F_N)}} \qquad (3)$$

Where Pr(a) is the probability of a successful classification (accuracy) and Pr(e) is the probability of a successful classification due to chance.

### *A. Binary Classification*

To compare the effectiveness of our implemented DCNN architectures in solving the binary classification problem, each model was trained upon our training dataset, which contained 23,408 images of dimensions 224 x 224 x 3. It was important to train each model over large number of epochs, in order to determine the number of training epochs required to optimise model performance. Consequently, each model was trained over 150 epochs, and evaluation was performed after 5, 10, 20, 30, 50, 75, 100, 125, and 150 epochs using our validation dataset which contained 2931 images. Throughout training, a standard 70:30 training/testing split was adhered to. We found that all models peaked in terms of performance very early in the training procedure. This is because a binary classification is relatively simple compared to many-class problems such as the ILSVRC, so our models quickly learn the patterns required to classify fire in images and overtraining occurs very easily. We used the results from this experimentation to train each model to the point at which the accuracy was maximised. A more in-depth statistical analysis of each model was then performed, the results of which are shown in Table 1.

**Table 1.** A full statistical analysis of each model after optimum training.

| Model | TPR | FPR | Accuracy | Precision | F | Kappa | MCC |
|---|---|---|---|---|---|---|---|
| AlexNet | 90.9 | **7.3** | 91.7 | **94.6** | 92.7 | 83.3 | 83.0 |
| GoogLeNet | 95.5 | 9.4 | **93.4** | 93.5 | **94.5** | **86.9** | **86.5** |
| VGG 13 | 93.3 | 11.4 | 91.3 | 92.0 | 92.7 | 82.7 | 82.1 |
| BinNet | 92.2 | 9.4 | 91.5 | 93.3 | 92.7 | 83.0 | 82.6 |
| GoogLeNet-2 | **96.0** | 10.3 | **93.4** | 92.9 | 94.4 | 86.8 | 86.3 |

Table 1 shows that GoogLeNet is the superior model for solving this binary classification problem, with the greatest F-Measure, Kappa, and Matthews correlation coefficient, which are the statistics that provide the best summary of overall performance. However, the performance of GoogLeNet-2 very closely matches that of GoogLeNet. To directly compare the strength of our GoogLeNet solution to previous solutions to this problem we trained this model upon the evaluation dataset created by Steffens *et al*. (Steffens et al. 2015). In order to perform this training process, we used 90% of the 20593 images – giving us a training dataset comprised of 18534 images which were resized to dimensions of 224 x 224 x 3. The remaining 10% of the original images were used to construct a validation dataset which comprised of 2059 images. We trained GoogLeNet for 28 epochs using these 18534 images. The results of this evaluation are presented in Table 2, along with the results obtained by training and evaluating the current state of the art solution (Chenebert et al. 2011) upon the same dataset. The statistical measures used by Steffens *et al*. which have not already been introduced in this work are the negative prediction value (NPV) (Eq. 4), false discovery rate (FDR) (Eq. 5), true negative rate (TNR), and false negative rate (FNR).

$$NPV = \frac{T_N}{T_N + F_N} \qquad (4) \qquad\qquad FDR = \frac{F_P}{F_P + T_P} \qquad (5)$$

**Table 2.** A comparison of our binary classification solution and the previous state of the art solution.

| Method | TPR | TNR | Prec | NPV | FPR | FDR | FNR | Accuracy | F | MCC |
|---|---|---|---|---|---|---|---|---|---|---|
| Chenebert | 0.990 | 0.724 | 0.857 | 0.979 | 0.275 | 0.142 | 0.009 | 0.890 | 0.919 | 0.773 |
| GoogLeNet | **1.000** | **0.999** | **0.999** | **1.000** | **0.001** | **0.001** | **0.000** | **1.000** | **1.000** | **0.999** |

Table 2 shows that our solution vastly outperforms the current state of the art solution to this problem, particularly in terms of false positive classifications. In fact, our solution performed only 1 incorrect classification out of 2059 unseen validation images.

### *B. Model Simplifications and Speed*

To design BinNet and GoogLeNet-2, we tested the performance of the models created by removing certain components from AlexNet and GoogLeNet. These models were trained using 25% of our training dataset, and evaluated upon our evaluation dataset. The original GoogLeNet architecture contained 9 Inception modules; we evaluated the performance of this architecture with varying numbers of Inception modules. Figure 6 (a) shows the number of parameters present in each of these models plotted against the maximum accuracy achieved by the model, with the label of each data point corresponding to the number of Inception modules present in the architecture. When removing layers of AlexNet, we found that

removing the first convolutional layer lead to a huge increase in parameters, so this was avoided. Figure 6 (b) shows the number of parameters present in each modified AlexNet model, plotted against the maximum accuracy achieved. Model *a* was produced by removing layer 3, *b* by removing layers 3 and 4, *c* by removing layers 3, 4, and 5, *d* by removing layer 6, *e* by removing layers 3, 4, and 6, and *f* by removing layer 2. The standard AlexNet architecture is represented by *g*.

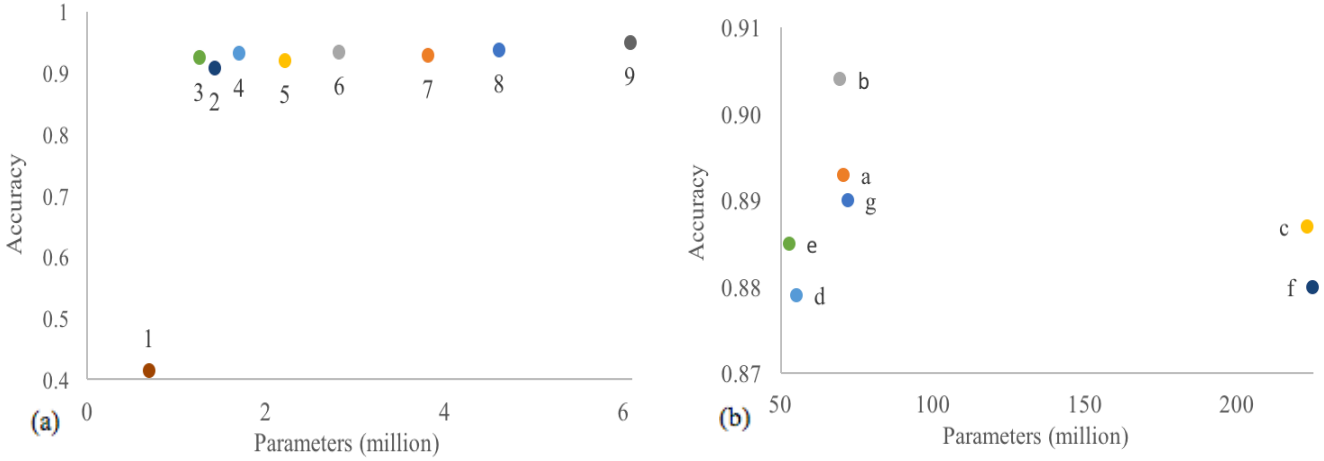


**Figure 6.** Parameter and accuracy comparison of GoogLeNet variations (a), and AlexNet variations (b).

Figure 6 (a) shows that accuracy tends to slightly decrease as the number of Inception modules decreases, whereas the number of parameters decreases significantly. The exception to this is the variation containing only 1 Inception module, which performs much worse. The GoogLeNet architecture containing 3 Inception modules is the variation with the fewest parameters which retains performance, and so this variation was used for GoogLeNet-2. Figure 6 (b) shows that model *b* improves upon the accuracy of all other variations, whilst containing fewer parameters than the original model, and so this variation was used as the basis for BinNet.

To evaluate the effectiveness of BinNet and GoogLeNet-2 against that of the existing architectures, we investigated the speed of each solution. This was done by performing live classification of a video feed of dimensions 608 x 360 x 3. This evaluation was performed using a 2.7 GHz Intel Core i5 processor, and 8GB of RAM. For each model the classification of 100 frames was performed, from which the average time taken to process each frame was recorded, allowing the calculation of the number of frames processed per second (FPS) by each model. Table 3 shows the accuracy of each trained model, along with the number of model parameters, the time taken to classify a single input frame, and the FPS achieved.

**Table 3.** Accuracy of conventional and simplified architecture, along with number of parameters (millions), the time taken to classify a single frame, and the frames classified per second, for AlexNet and GoogLeNet.

| Model | Parameters (m) | Accuracy (%) | Time/Frame | FPS |
|---|---|---|---|---|
| AlexNet | 71.9 | 91.7 | 0.247 | 4.047 |
| BinNet | 68.3 | 91.5 | 0.059 | 17.017 |
| GoogLeNet | 6.1 | 93.4 | 0.389 | 2.568 |
| GoogLeNet-2 | 1.2 | 93.4 | 0.119 | 8.368 |

Table 3 shows that the accuracy of BinNet is only slightly worse than that of AlexNet, yet it can perform a classification 4.2 times faster. Also, the results show that GoogLeNet-2 achieves the exact same accuracy as GoogLeNet, but can perform a classification 3.3 times

faster. These results show that our model simplifications have been successful in greatly reducing model complexity, and therefore increasing classification speed, whilst preserving model performance in our problem.

### *C. Localization*

Due to the fact that GoogLeNet-2 was the superior model in our binary detection solution after considering performance and speed, it was used exclusively in our localization solution. In order to train GoogLeNet-2 to perform localization, we extracted 90% of the frames from the dataset of Steffens *et al*., in order to allow us to evaluate this approach upon the remaining 10% of the frames. We chose a maximum number of 100 superpixels per image because this enabled a good segmentation of each image, whilst keeping computation minimal. After extracting superpixels from the training images, we were left with 54,856 positive, and 1,674,002 negative example of superpixels. This was far too much data to train in a reasonable time with the hardware we had access to, so we eliminated 90% of the negative superpixel examples, to give a more manageable 167,400 negative examples. Figures 7 and 8 illustrate the performance of our solution across a range of input images. Our solution produces a green contour around superpixels which are classified as fire and a red contour otherwise.

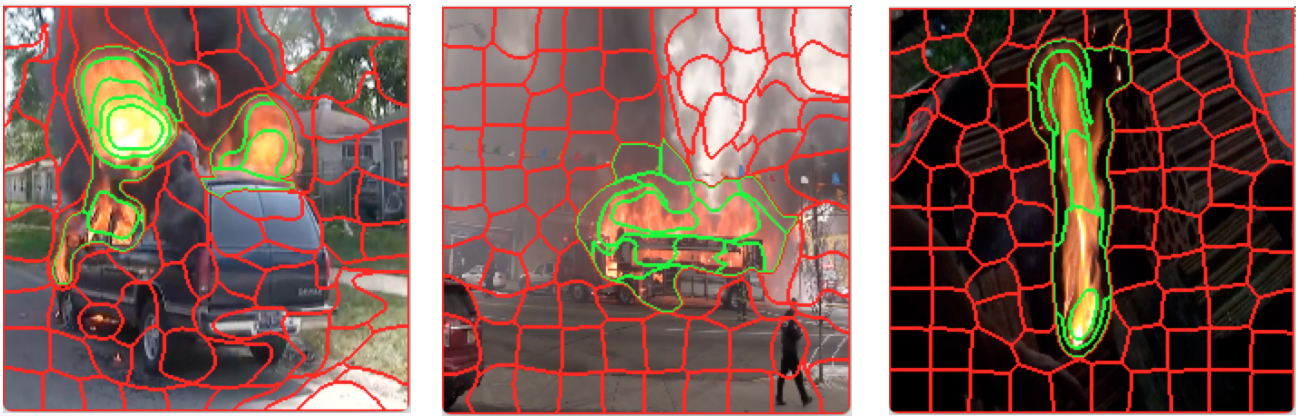

**Figure 7.** True positive classifications of fire within input images.

Figure 7 demonstrates the strength of our solution, illustrating the correct localization of fire within a range of images which each feature fire in different environments.

(a) (b) (c)

**Figure 8.** A false negative detection (a), a partial detection (b), and a false positive detection (c).

Figure 8 illustrates the limitations of our solution: (a) shows a false negative detection, (b) shows only partial detection of fire, and (c) shows the false positive detection of fire.

To evaluate the performance of our localization solution, we converted our contour-based output into a bounding box output by constructing a box which bounded all fire-classified superpixels. OpenCV provided an easy way to perform this conversion. These bounding boxes enabled the comparison of our results to the ground truth bounding box labels of Steffens *et al*. A detected bounding box is said to be correct if the ratio of the intersect of the detected box and the ground truth box, to the ground truth box, is greater than 0.5. This ratio is called the *similarity* (S) between the detected box and ground truth box. The remaining 10% of the dataset of Steffens *et al*. which was not used to train the classifier was used for validation, comprising of 1178 positive examples and 881 negative examples. Table 4 (a) shows the results of our localization solution compared to the current state of the art solution in terms of detection, whereas (b) shows the same comparison but in terms of the location-based metrics presented by Steffens *et al*.

**Table 4.** A comparison of our localization solution and the previous state of the art solution in terms of detection (a), and location-based metrics (b).

(a)

| Method | TPR | TNR | Prec | NPV | FPR | FDR | FNR | ACC | F | MCC |
|---|---|---|---|---|---|---|---|---|---|---|
| Chenebert | **0.990** | 0.724 | 0.857 | **0.979** | 0.275 | 0.142 | **0.009** | **0.890** | **0.919** | **0.773** |
| Dunnings | 0.924 | **0.835** | **0.882** | 0.891 | **0.165** | **0.118** | 0.076 | 0.886 | 0.902 | 0.766 |

(b)

| Method | Prec | TPR | F | S |
|---|---|---|---|---|
| Chenebert | 0.832 | **0.979** | **0.902** | **0.801** |
| Dunnings | **0.843** | 0.924 | 0.882 | 0.779 |

Table 4 clearly shows that, overall, the method of Chenebert *et al*. outperforms our localization solution. However, our solution performs almost as well, with results closely matching, or sometimes improving upon, those of Chenebert *et al*. The speed of this solution was evaluated using the same method that was used to evaluate the speed increase of our model simplifications, and produced an average classification time of 6.372 seconds, which equates to 0.16 FPS.

## V. Evaluation

In this section, we evaluate the strengths and limitations of our solution, reflecting upon the original research question: *Can deep neural networks be used for the task of detecting and localizing fire within video imagery in order to advance the current state of the art?*.

### *D. Solution Strengths*

We have demonstrated through the application of various current state of the art architectures that DCNN perform excellently in the task of visual fire detection. We showed that DCNN can be used to solve the binary classification problem of determining whether fire is present in a given image with an accuracy of 93.4%. This is a vast improvement upon the current state of the art (Chenebert et al. 2011), which attained an accuracy of 87.3%. Our solution was evaluated upon a set of 6 videos which had not been used in the training of our models, each containing different types of fire – showing that our solution is robust. Additionally, the fact that our solution can perform classification of images of any dimension further increases

its flexibility. When evaluating both our solution and the current state of the art solution upon the dataset of Steffens *et al*., a visual fire detection evaluation dataset, we saw that our solution outperformed that of Chenebert *et al*. in every aspect. Most importantly, our solution achieved an accuracy of 100% compared to the 89% of Chenebert *et al*.

We have also demonstrated that a superpixel segmentation of an image can be used in conjunction with our binary classification solution to perform localization of fire within the image. Our localization solution produced a detection accuracy of 88.6%, which almost matches the 89% accuracy achieved by Chenebert et al, and a location-based F-Measure of 0.882, almost matching the 0.902 of Chenebert *et al*.

In an effort to decrease the complexity of existing DCNN architectures whilst preserving their performance in our problem, we have produced two new architectures which achieve this aim. Our BinNet model, which was constructed using principles from AlexNet, achieves an accuracy only 0.2% less than that of AlexNet, yet performs 4.2 times faster in our problem. Additionally, our GoogLeNet-2 model achieves an accuracy equal to that of GoogLeNet, yet performs 3.3 times faster. In producing these models, we have shown that current state of the art DCNN architectures, which are designed to perform well in 1000-class problems, can be simplified to solve binary problems without a loss of performance. This leads to the conclusion that additional complexity and depth in DCNN, which has been used to improve performance in many-class problems, does not necessarily improve performance in binary problems such as ours. These simplifications allowed our optimum solution to process 8.4 FPS, rather than 2.6 FPS, which enables a much smaller delay in the detection of fire. The speed of our solution also improved upon the state of the art; Chenebert *et al*. is capable of processing 12 FPS, whereas our BinNet solution is capable of processing 17 FPS whilst achieving an accuracy approximately 4% greater than that of Chenebert *et al*.

### *E. Solution Limitations*

The main limitation of our work is that, unlike our binary classification solution, our localization solution does not improve upon the current state of the art solution to this problem. This limitation was most likely caused by the fact that we used bounding box labels to produce ground truth labels for each superpixel. This lead to superpixels which did not actually contain fire being labelled as containing fire, because they fell within the bounding box label for an image. Rectifying this limitation would involve producing a dataset of contour-labelled images which would be very time consuming. The speed of our localization solution is very limited, with 0.16 FPS compared to the 12 FPS of Chenebert *et al*., which limits the use of this solution in autonomous agents as they require real-time interaction with their environments. However, our superpixel segmentation method was implemented using Python, when there are implementations such as gSLICr (Ren et al. 2015) which could be used to perform this process much faster. This would also allow for each image to be segmented into more superpixels, which would likely increase performance.

### *F. Approach*

Our approach of first producing a solution to the binary classification problem of detecting fire within video imagery, and then extending this to produce a localization solution, eliminated the risk of having no solution if localization performed poorly. Also, this structure allowed for the constant training and evaluation of model architectures throughout the duration of the project. This was an agile development approach – a basic solution was adopted and implemented, and then iteratively improved, allowing us to identify problems and address them with ample attention. A great portion of the time spent on this project was

devoted to the collection and manipulation of data. Using datasets established by previous solutions to this problem in the early stages of work would have been a benefit, as there was a great deal of trial and error involved with labelling, formatting, and transporting data. In addition to this, only a small portion of our entire dataset was ever used for training, so time was wasted in compiling and labelling this superfluous data.

In the early stages of this project, decisions were made concerning the implementation tools that would be used. Using TensorFlow proved to be a challenge, and introduced a sharp learning curve. Also, many of the latest state of the art techniques could not be implemented in TensorFlow, such as region-based convolutional neural networks (Girshick et al. 2016), which would likely have proved to be a better localization technique than ours. More investigation into the capabilities of different tools would be carried out in the early planning stages if this project were to be undertaken again.

These minor shortcomings in the planning of this project did not, however, have an adverse effect on the quality of the solution produced. Our solution produces results which have been shown to improve upon the current state of the art in visual fire detection by applying DCNN. In addition to this, we have shown that simplifications to complex existing DCNN architectures can be made to improve speed whilst preserving performance in binary problems such as this.

## VI. Conclusion

In this project, we have produced a solution to the visual fire detection which achieves an accuracy of 93.4%, improving upon the 87.3% of the current state of the art solution to this problem (Chenebert et al. 2011). To achieve this, three current state of the art DCNN architectures (Krizhevsky et al. 2012; Szegedy et al. 2015; Simonyan & Zisserman 2015) were implemented, and their performance in this problem evaluated. Ideas and principles were taken from the design of these architectures and used to create two new, simpler, architectures. These architectures closely matched the performance of the state of the art architectures when applied to our problem, but performed classification more than three times faster. Our results demonstrate that state of the art DCNN architectures can be greatly simplified, yet still retain performance in binary problems such as ours.

We also extended this solution to perform localization of fire within video imagery by producing a superpixel segmentation of each image, and classifying each superpixel individually. This solution was shown to produce results which were comparable to those of the current state of the art solution.

We demonstrate that DCNN are currently the primary candidate for producing a visual fire detection solution which is reliable enough for widespread implementation within the real world, when used in conjunction with existing fire detection methods. Future work may investigate the effectiveness of newer state of the art DCNN architectures (He et al. 2015) when applied to this problem, as this is a rapidly advancing field with stronger architectures being introduced year after year. Additionally, expansion of our localization solution could be conducted if a contour-labelled database were to be constructed for use in this problem, or a faster superpixel segmentation method was applied (Ren et al. 2015). It is also likely that modern techniques for performing object localization such as region-based convolutional neural networks (Girshick et al. 2016) will outperform our superpixel-based localization solution when applied to this problem.

## REFERENCES


Achanta, R. et al., 2012. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11), pp.2274–2281.

Ahuja, N., 2004. Vision based fire detection. *Proc. International Conference on Pattern Recognition*. 4(1), pp.134–137, Vol.4.

Bland, J.M., 2008. Measurement in Health and Disease: Cohen's Kappa. *University of York Department of Health Sciences*, pp.1–11.

Çelik, T. & Demirel, H., 2009. Fire detection in video sequences using a generic color model. *Fire Safety Journal*, 44(2), pp.147–158.

Chen, T.-H., Kao, C.-L. & Chang, S.-M., 2003. An intelligent real-time fire-detection method based on video processing. *Proc. International Carnahan Conference on Security Technology*, (q), pp.104–111.

Chenebert, A., Breckon, T.P. & Gaszczak, A., 2011. A Non-temporal Texture Driven Approach to Real-time Fire Detection. *Proc International Conference on Image Processing*, pp.1741–1744.

Deng, J. et al., 2009. ImageNet : A Large-Scale Hierarchical Image Database. *IEEE Conference on Computer Vision and Pattern Recognition*, pp.248–255.

Foo, S.Y., 1996. A rule-based machine vision system for fire detection in aircraft dry bays and engine compartments. *Knowledge-Based Systems*, 9(8), pp.531–540.

Girshick, R. et al., 2016. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1), pp.142–158.

Haralick, R.M., Shanmugam, K. & Dinstein, I., 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man and Cybernetics, Vol. 3, No. 6, pp. 610–621.

He, K. et al., 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385.*

Ko, B.C., Cheong, K.-H. & Nam, J.-Y., 2009. Fire detection based on vision sensor and support vector machines. *Fire Safety Journal*, 44(3), pp.322–329.

Krizhevsky, A., Sutskever, I. & Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pp.1–9.

LeCun, Y. et al., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pp.2278–2323.

Le Cun, Y. et al., 1990. Hand-Written Digit Recognition with a Back-Propagation Network. *Advances in neural information processing systems*, pp.396-404.

Morerio, P., Marcenaro, L., Regazzoni, C. S., Gera, G., 2012. Early fire and smoke detection based on colour features and motion analysis. *IEEE International Conference on Image Processing*, pp.1041–1044.

OpenCV Open Source Computer Vision and Machine Learning Library. [Online; accessed 25th April 2017]. Available at: http://opencv.org/about.html

Phillips, W., Shah, M. & da Vitoria Lobo, N., 2002. Flame Recognition in Video. *Pattern Recognition Letters*, 23(1–3), pp.319–327.

Ren, C.Y., Prisacariu, V.A. & Reid, I.D., 2015. gSLICr: SLIC superpixels at over 250Hz. *arXiv:1509.04232*.

Russakovsky, O. et al., 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3), pp.211–252.

Scikit-Image Python Image Processing Library. [Online; accessed 29th April 2017]. Available at: http://scikit-image.org

Simonyan, K. & Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, pp.1–14.

Steffens, C.R., Rodrigues, R.N. & Silva da Costa Botelho, S., 2015. Non-stationary VFD Evaluation Kit: Dataset and Metrics to Fuel Video-Based Fire Detection Development. *12th Latin American Robotics Symposium and Third Brazilian Symposium on Robotics*, pp.135–151.

Szegedy, C. et al., 2015. Going deeper with convolutions. *arXiv:1049.4842.*

Tensorflow Open Source Machine Intelligence Library. [Online; accessed 25th April 2017]. Available at: http://tensorflow.org

Toreyin, B.U., Dedeoglu, Y. & Cetin, a E., 2005. Flame detection in video using hidden Markov models. IEEE Conference on *Image Processing*, 2, pp.1230–1233.

YouTube Video Sharing Website. [Online; accessed 25th April 2017]. Available at: http://youtube.com

Zhang, D.Z.D. et al., 2009. Image Based Forest Fire Detection Using Dynamic Characteristics with Artificial Neural Networks. *International Joint Conference on Artificial Intelligence*, (2), pp.290–293.