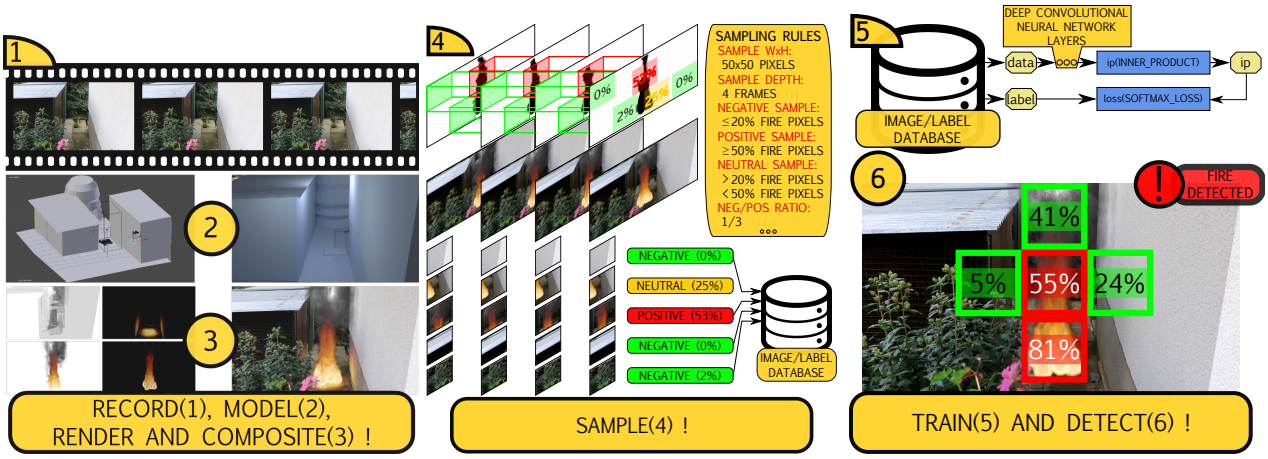
Excel@FIT 2015
http://excel.fit.vutbr.cz

# Detection of Fire in Images and Video

Bc. Tomáš Poledník*




**Abstract**

This paper deals with fire detection in image and video by machine learning, specifically deep convolutional neural networks, using Caffe framework. The aim is to create a vast set of testing data that could be used as the base element of machine learning detection and create a detector usable in real-time application. For the purposes of the project a set of tools for fire sequences creation, their segmentation and automatic labeling is proposed and created together with a large test set of short sequences with artificial modelled fire.

**Keywords:** Fire detection, Image processing, Video sequence, Machine Learning by deep convolutional neural networks, Computer vision, Caffe, Fire modelling, Fire scene compositing

**Supplementary Material:** Demonstration Video — Downloadable Code And Examples

*xpoled06@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

This project serves as an alternative method to ordinary fire detection using short-range smoke and heat sensors. No special hardware is required here, just a camera and a computer analysing the camera's output. In case of perfect results, such system could be used in everyday life. The main goal is to try a modern way of detection in image and video based on machine learning with deep convolutional neural networks (DCNNs). As DCNNs are new in this field and are considered extremely powerful (for instance thanks to the results from the work of Krizhevsky et al. [1]), it would be interesting to see how much the results differ from the fire detections of other authors. Fire is not an easy thing to record, so finding a way to create such records artifically is another motivation. It is therefore a challenge — creating a real fire detector that never saw a real fire.

The problem can be divided into two core tasks. The first one involves creating a large dataset of fire and non-fire samples and using these to train a model of deep convolutional neural network. The second task is to make a software detector that analyses image (video) input, tests its content on the pretrained model and returns a result. The bonus is represented by the multipurpose usability of fire-creating and sampling tools as they can be used in some other similar project

and are not limited to this one. The proper solution would be a detection that finds fire in a video, if present, as soon as possible — within 1 to 10 seconds. Image analysis, which is implemented in this work, does not include temporal information and represents a part of video detection — my personal goal is for the number of true positives in images to exceed 95%. The evaluation is always done per image where, in case of fire, at least one fire area must be marked with a probability higher than 50%.

There are many methods for fire detection in image and video, each with its own advantages and disadvantages. This paragraph contains the summary of different approaches to the detection. Every method's detection is based on one or more visible qualities of fire. These include its colour, shape, motion, frequency, spatial change and smoke generation. Colour is the most dominant quality used for detection. Chen et al. [2] detect colour properties of fire described in Qunitiere's book [3]. The main property is fire's transition from red, through yellow and up to white colour. As RGB's components do not carry the necessary information about the colour's intesity, Çelik and Demirel [4] incorporate YCbCr model into their method. They use statistical analysis of fire's colour trained by their test set. Colour presents valuable information but it cannot be used solely as many objects share the same colour as a real fire. Fire's colour, shape and spatial change are the detected features in the work of Liu et al. [5]. Liu et al. observed that each of fire's colours can be distiguished by its typical relative position. The flame's centre (core) is always the brightest spot. Moving from the centre this colour becomes yellow, orange and red. The shape of every fire region is given by its border. This border is represented by one-dimensional signal composed out of $N$ points — pixels of fire region's border. Fire's shape is then described using a Fourier Descriptor. In every single frame Liu et al. calculate the coefficients of Fourier series of the border of every fire region. After that it is observed how these coefficients change in time. Liu et al. presume fire's basic shape — its low-frequency components — stays the same. However, high coefficients will exhibit large change and therefore can be detected. Recognition of fire's shape and spatial change with this method is extremely accurate (authors state 99% accuracy), however, it fails in case of partly visible fire border. A trained colour model, frequency scanning, spatial change and motion analysis are used for detection in the work of Töreyin et al. [6]. Precalculated distribution of possible fire colours in RGB model space is compared to the colour of each pixel. The decision model is created using a combination of several pretrained normal distributions. Spatial wavelet analysis is used for frequency scanning. It can detect flickering (oscillation) of fire pixels. For the analysis to be able to capture a pixel's frequency between 1 and 10 Hz, the video's frame rate must be at least 20 frames per second. In case this frame rate cannot be reached, the analysis may fail to detect the correct frequency. The spatial change monitoring is performed using wavelet analysis. Spatial change is caused mainly by turbulent and random motion of pixels which is one of fire's features. Any other object should exhibit only very subtle or no random motion. Spatial change and frequency analysis differentiate fire well but only work without too much noise in the input or when the fire is not too distant from the camera. Moving regions are detected using a modified method of background subtraction. Çelik et al. [7] uses a pretrained colour model and also detects fire's motion by creating and updating a model of background. Video's background is modelled using normal distribution. Using such real-time models is quite computationally expensive and requires some time to initialize their parameters. However, later on they can adapt to changes in the environment. Lastly, there is fire's smoke generation. Detection of gray smoke by image separation is a part of the work of Tian et al. [8]. A background model is constructed and the smoke is detected by its gray colour and partial transparency. Recognition of smoke can influence fire detection but, as sometimes the smoke is almost absolutely transparent, such detection might fail.

Deep learning using deep convolutional networks presents a modern way of detection today. There are different tools that allow working with these network models. There is Theano [9] for *Python*, Torch7 [10] built on *Lua* and Caffe Deep Learning Framework [11] for *Python* and *C++*. As Caffe presented the fastest results in deep learning with deep convolutional neural networks at the beginning of my work and I used *C++* for the development, I chose Caffe.

A short overview of my solution follows. The first part that needed to be dealt with was the fire data creation. I recorded several static scenes with a camera. The best way I found to gain realistic fire effect was to use a 3D modelling tool, specifically Blender[1]. With it, I modelled the scene with a burning fire, rendered it and pasted it onto the frames of the recording. My solution involves experiments based on two approaches to created fire sequences and their initiation into a model's training — per image and per segment and the comparison of their results. I created a sampler that

[1] Blender Online Community: http://www.blender.org/

was used to sample the fire recording according to a given approach. The model was created, trained and tested using Caffe [11].

To sum up, I created a way that allows anyone who is at least a little experienced in 3D modelling to create a fire animation, sample it and use it for a fire model's training. The compositor and sampler present applications that can even be used in separate projects. As for the detection so far, I experimented only on images. The detector's per segment approach trained only on modelled fire showed that it can be used for real fire detection with 50 out of 50 correct detections. In the future work more data will be created and more varied scenes will be used for fire creation. Video detection is also a part of the future work.

## 2. Fire Video Rendering and Composition

The proposed method of fire detection using convolutional neural networks requires a vast number of testing data. Each testing sample must be a short sequence of fire burning in a real-life environment. Using the techniques described below I created 2000 frames of fire animation footage which can be used for detection testing.

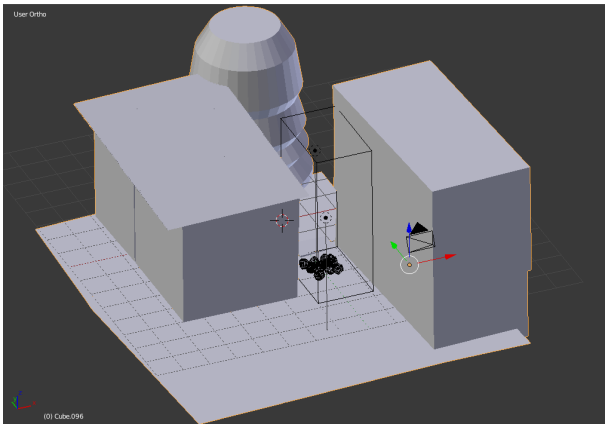
### 2.1 Fire Video Sequences Creation

The main disadvantage of fire detection is that fire on its own represents a not so common phenomenon. Its size, shape, dynamics and colour are very variable. It is not an easy task to find an extensive quality resource for fire images let alone fire animations. In this paper I used a different approach to normal video creation.

Instead of filming a real-life fire, a normal scene without fire was filmed with a static camera. The camera's resolution was $1920 \times 1080$ pixels. For every scene, its low-poly 3D model is created in Blender version 2.71+. The notion is to create a fire simulation in the modelled environment (an animation of burning fire) and render it using Blender's Cycles ray tracing renderer. Then extract it, together with all of its lighting effects and shadows it creates in the scene, and paste all of these onto the frames of the filmed scene.

Creating my own 3D modelled fire has the advantage of total control over its appearance, behaviour (physical simulation) and its lighting effects on the environment. There is also no problem in changing the camera view.

I have filmed a total of 10 static scenes that capture different kinds of environments of both exterior and interior. I chose 2 of these scenes and created their correspoding 3D models. An illustration of the created



(a)

(b) (c)

**Figure 1.** Demonstration of the created Scene 1 and its comparison to the original image: a - scene's 3D model view in Blender, b - original image used as modelling template, c - rendered scene in the same view as the original captures.

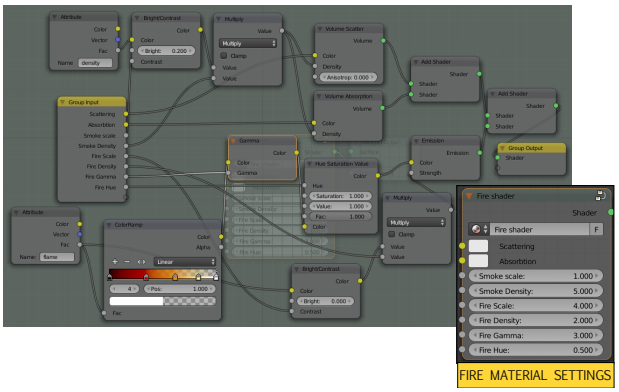
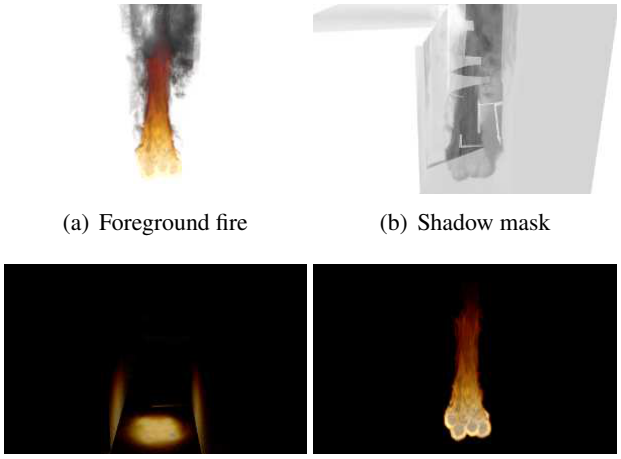



**Figure 2.** Fire material node setup in Blender.

outdoor scene, Scene 1, can be seen in Figure 1.

To add fire to this scene, Blender's fire simulation and physics engine are used. I have created my own adjustable fire material — fire shader. The setup for this material in Blender and its settings can be seen in Figure 2. Fire is simulated at multiple locations in the scene model and a different random seed for the simulation is always used. I added blowing wind to the scene completely randomly for a more realistic effect of a burning fire. This guarantees that fire animations never look the same.

The fire scene data is rendered into multiple images — fire and smoke foreground, scene's shadows,



(a) Foreground fire (b) Shadow mask

(c) Light mask (d) Pixel mask

**Figure 3.** Display of rendered data from a random sequence frame.

scene's fire light emission — and is later composited into images of a sequence. Part of the rendering process is the creation of a fire pixel mask which is used as an annotation for the detection. All of the Blender outputs are shown in Figure 3.

The scene models and scripts needed for batch rendering can be found on the supplementary material website.

### 2.2 Fire Sequence and Real-Life Scene Composition

Rendered fire animations from Section 2.1 need to be composited with the frames of a filmed scene. To have as much control over fire images (animations) creation as possible, I programmed my own compositor using the OpenCV library [12]. I created it universally so it can be used for different kinds of compositing and not just for the creation of fire images.

Using the rendered fire sequence from Section 2.1 and my compositor the final fire video can be created. The process of a single fire frame creation and compositing can be seen in Figure 4 with numbered steps $1-5$.

1. In the top right corner there is the original image extracted from a video sequence. This becomes the compositor's background.
2. At first, this image is masked to forbid the compositor to change the areas that are not supposed to change in the video. In case of the example image in Figure 4 this mask is used on the bush which is very close to the camera in the extracted frame.
3. Masking is followed by multiplication with a shadow mask. Shadows, which are stored in a grayscale image, darken parts of the frame.
4. After that fire light emission mask is added that lights up pixels just like a real fire would when burning. Simple addition is used for this step. Emission masks should be mainly used on scences which are poorly lit. Adding them to a sunny scene would be counterproductive.
5. On top of that an image containing the rendered fire is added using alpha-blending. This image preserves alpha channel and acts as a simple overlay.

All of these components combined create a fire frame. A short sequence of these fire images together with their corresponding fire pixel masks serve as a single testing sample.

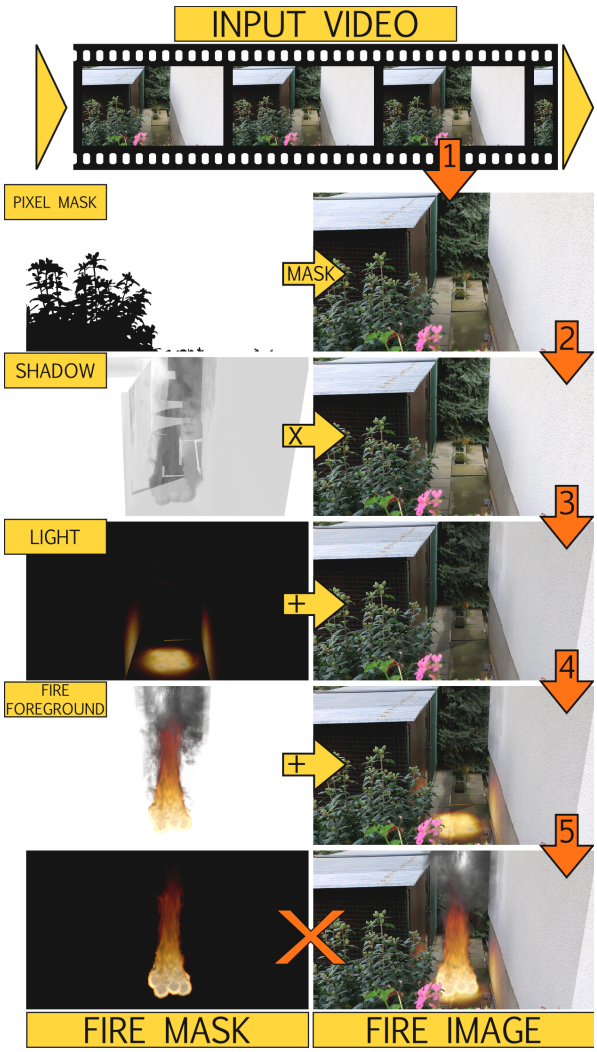



**Figure 4.** Depiction of test image creation: 1 - video frame image extraction, 2 - pixel mask application, 3 - shadow mask multiplication, 4 - emission mask addition, 5 - fire foreground addition (= fire image generation), X - fire mask and fire image output.

Figures 5 and 6 show examples of composited fire images from different scenes.

(a)

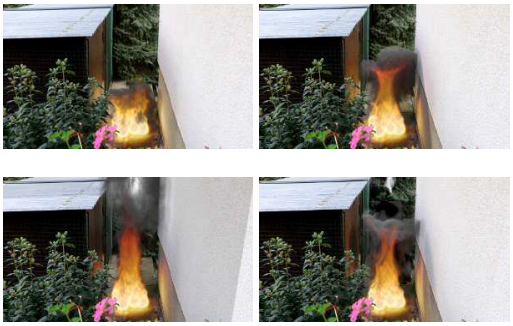


**Figure 5.** Examples of composited fire images from Scene 1: a - original video frame image before composition.

(a)

**Figure 6.** Examples of composited fire images from Scene 2: a - original video frame image before composition.

The compositor source code files and complete manual to using the interface can also be found on the supplementary material website.

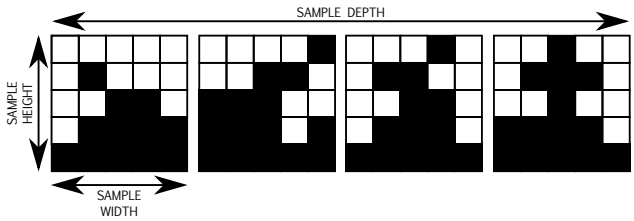



**Figure 7.** Example of a $5 \times 5 \times 4$ segment represented by its labels (black = positive (fire) pixel, white = negative (non-fire) pixel).

## 3. Training Fire Samples Creation

The resulting fire sequences from Section 2.2 serve as training and testing data for the detector. The detector is supposed to learn patterns and parameters from its input and recognise these during fire detection. I propose three different approaches to the detector's training - per the entire image, per pixel and per segment. Per pixel and per image are special cases of per segment approach. For video analysis, every sample may have its depth that corresponds to a certain number of frames. To sample the data in these ways I created a separate sampler using the OpenCV library [12].

The sampler takes 2 sequences of images as input — fire images and pixel labels of these images. The required labels are binary masks (black and white) that can be generated from fire masks from Figure 4 using my compositor. A starting frame is chosen from this sequence and a subsequence of a set sample depth beginning with this frame is analysed for samples. An example of a sample with dimensions of $5 \times 5 \times 4$ from such a subsequence of labels is shown in Figure 7.

The sampler contains different settings that influence the generation of samples. These include:

- sample dimensions,
- required percentage of fire pixels in a sample's volume to be negative/positive,
- the maximum number of required negative or positive samples,
- percentages of maximum number of samples within a subsequence,
- random or constant step between sequence's frames when picking candidates for subsequences' first frames,
- random or constant step between filtered samples when picking candidates from sample lists,
- positive/negative samples ratio (used within a subsequence),
- postprocessing options (e.g. resizing) after generation.

An example of one iteration of the sampling process can be seen in Figure 8.

1. In the left part of Figure 8 there is an example of a fire sequence that contains both the labels and the composited fire images. This sequence is traversed and a starting frame of a subsequence of given sample depth is chosen according to the sampler's settings (e.g. randomly).
2. The subsequence's frames are selected and analysed separately.
3. Lists of all negative and positive samples available on the given subsequence are generated (neutral samples are discarded) according to the sampling rules (e.g. all segments with fire pixels volume above 50% are put into the positive samples list). These lists are then filtered according to the current settings, e.g. the required ratio of positive/negative samples and their maximum number.
4. All the samples that passed the filtration are stored.

The sampler source code files and a manual to using the interface can also be found on the supplementary material website.

## 4. The Fire Detector

I used Caffe Deep Learning Framework [11] for the implementation of the detection method.

The detector uses the same structure of the deep convolutional neural network that was proposed in the work of Krizhevsky et al. [1] for Imagenet classification challange [13]. In the Caffe environment, this net structure is referred to as Caffenet and it is shown in Figure 9. For images, this model is extremely versatile. For video detection in the future work it must be adapted. I made a few changes to the net's structure though. One was changing the input layer to the corresponding image dimensions when conducting experiments. As my aim is to find fire in an image, there are only two labels in my dataset — fire and non-fire image (segment). As Caffenet was originally created for classification task that involved hundreds of classes, the training process would be a lot longer without this change.

For image detection I completely omitted the per pixel approach as the necessary training process was extremely lengthy and a single pixel does not include that much information. It will be a part of the future tests. Experiments test the detection's results of per $128 \times 128$-pixel and $256 \times 256$-pixel segment approaches and per image approach in single images. The convolutional net's input is an image of the specific size and its output is a number that represents the probability of classifying this image as fire.

I used 3 — training, validation and testing — sets of data for the experiments. For convolutional network's training I used the composited 3D fire images. One sample of per image approach corresponds to one unsegmented image. Every image that enters the net, when using this approach, must be resized in the first layer to a maximum of $256 \times 256$ pixels as larger resolutions cause extreme slowdowns and this size should capture just enough details. As the net's structure is created to accept any kind of image and resizes them during processing, any image of a varied size is considered a sample. For per image training I picked 900 fire images from the composited 2000. I picked 800 images from the rest of these for the model's validation. I omitted 300 images that looked quite similar to the already used ones. Then I collected 1280 real non-fire images of mostly city buildings from LabelMe[2] database. All of these images were of a varied size starting at $640 \times 480$ pixels. 780 of these are used for per image training (as non-fire samples) and 500 are used for validation.

For per segment approach I sampled the 900 training fire images and 780 non-fire images using these main options set for samples generation:

- sample depth is always equal to 1,
- positive sample has more than or equal to 50% of its area formed by fire pixels,
- negative sample has less than or equal to 20% of its area formed by fire pixels.

I generated 1300 fire samples and 1010 non-fire samples of $128 \times 128$ pixels. Then I generated the same amount of samples of $256 \times 256$ pixels. These were used for per segment training. For per segment validation, I sampled the images used for per image validation and acquired 800 fire samples and 800 non-fire samples of $128 \times 128$ and $256 \times 256$ segments respectively.

The described validation data is meant to measure the neural network's accuracy on a similar but not the same dataset as the training data.

The training set sizes together with the required training time for 3 models is shown in Table 1. The accuracy of these models tested on the validation set is presented in Table 2. The results show that per segment approaches are more capable of distinguishing modelled fire.

[2] LabelMe website: http://labelme.csail.mit.edu/

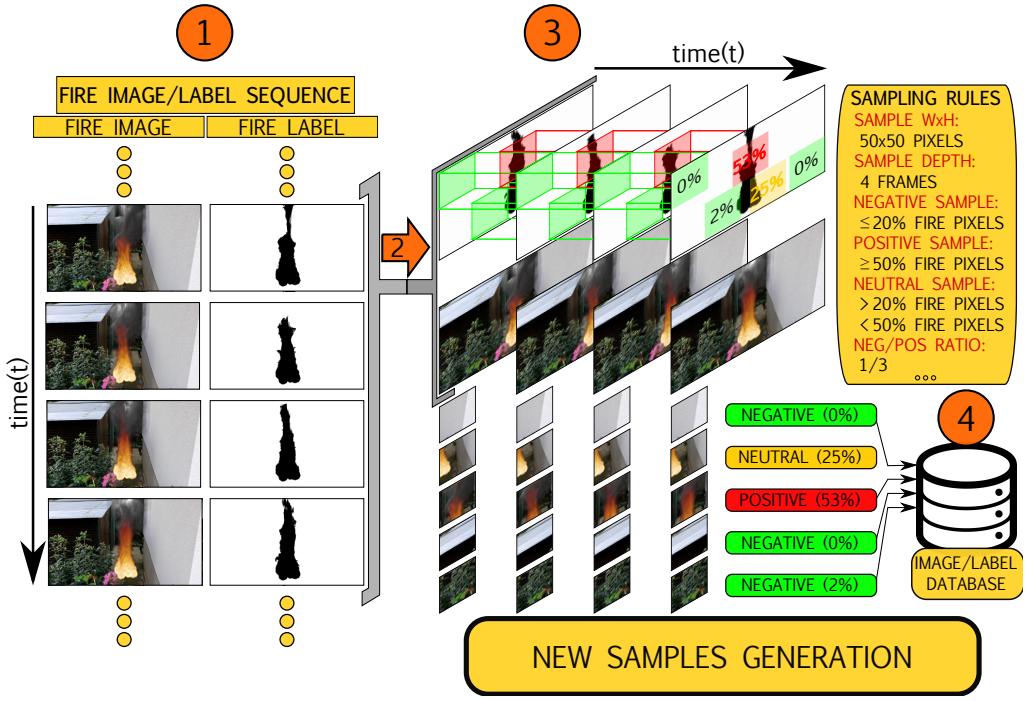



**Figure 8.** Depiction of samples creation: 1 - fire frames and labels traverse, 2 - sample frames selection, 3 - frame labels analysis according to the sampling rules and location of possible samples of given dimensions, 4 - positive and negative samples extraction and their storage in the database.

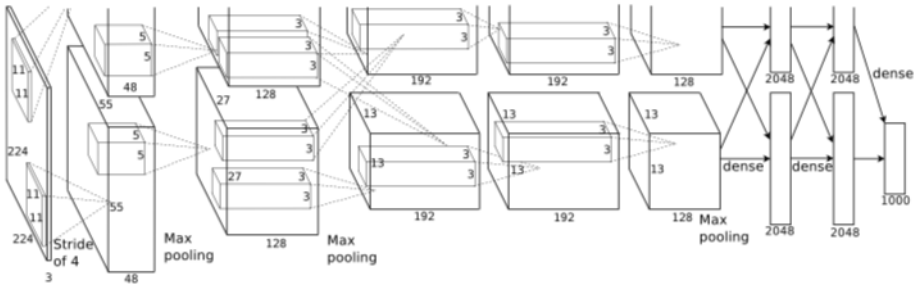



**Figure 9.** Deep convolutional neural network structure used in the work of Krizhevsky et al. [1] for Imagenet classification. Image adapted from article [1].

My goal is to create a detector of a real fire. Therefore, for real fire tests I used separate images downloaded from LabelMe which included 50 random real fire images and 50 non-fire images. For a true positive the detector must mark at least one visually undoubtable fire area with a probability higher than 50%. The results of these tests can be seen in Table 3.

The net's initial learning rate was set to 0.001. In case of a higher number (faster learning) the net's weights would reach too high numbers and the net would require a change in topology. Such changes — e.g. adding rectifier (ReLU) units to correct output weights — increased the processing time and were therefore undesirable. I set the number of training epochs (number of passes over the entire training data) to about 50 as higher numbers did not make a difference in validation accuracy. The only difference was that the training process took almost twice as much time. This might have been caused by the limited number of scenes that are very familiar and the net learns them too well.

As can be seen in Table 3, per image approach was the least successful. This was probably caused by many different details around the fire in an image that the classifier fails to distinguish. Per segment approach was far better scoring 100% true positives. Examples of the correct detector's per segment output on fire images can be seen in Figure 10. $256 \times 256$ segments sometimes failed to find fire in comparison to smaller segments. This can be seen in Figure 12. Smaller segments caused more false detections though which can be seen at the example of a fire station in

**Table 3.** Real fire image tests comparing the results of per image and per segment approaches. Left column contains the sizes of segments or images used as real input testing samples of an approach (per image data contains samples of variable size). Second and third column, Fire images and Non-fire images, present the numbers of these images used for testing. TP (true positives) presents the percentage of correctly detected fire in 50 real fire images (only stating that there is/is not fire in the image by marking at least one fire area with a probability higher than 50%). FP (false positives) gives the percentage of false detections in 50 non-fire images. Processing time is the time required to process one sample of Sample size from the input image.

| **Per segment** | | | | | |
|---|---|---|---|---|---|
| **Sample size** | **Fire images** | **Non-fire images** | **TP** | **FP** | **Processing time** |
| *128 × 128* | 50 | 50 | 100% | 24% | 0.11 seconds |
| *256 × 256* | 50 | 50 | 84% | 5% | 0.11 seconds |
| **Per entire image** | | | | | |
| *VARIED* | 50 | 50 | 6% | 2% | 0.20 seconds |

**Table 1.** Training information showing the comparison of per segment and per image approaches. Left column contains the sizes of segments or images used as input training samples (per image data contains samples of variable size). Fire and Non-fire samples present the numbers of these samples and Training time is the time (in hours) required to train the model. All the training fire samples contain only 3D modelled fire.

| **Per segment** | | | |
|---|---|---|---|
| **Sample size** | **Fire samples** | **Non-fire samples** | **Training time** |
| *128 × 128* | 1300 | 1010 | 4 hours |
| *256 × 256* | 1300 | 1010 | 4 hours |
| **Per entire image** | | | |
| *VARIED* | 900 | 780 | 5 hours |

**Table 2.** Validation results comparing the approaches of per segment and per image. Left column contains the sizes of segments or images used as input validation samples (per image data contains samples of variable size). Fire and Non-fire samples present the numbers of these samples and Accuracy is the accuracy of the model's classification on the validation data. All the validation fire samples contain only 3D modelled fire and are different from the training data. This table shows how well the models can classify artifical fire.

| **Per segment** | | | |
|---|---|---|---|
| **Sample size** | **Fire samples** | **Non-fire samples** | **Accuracy** |
| *128 × 128* | 800 | 800 | 98% |
| *256 × 256* | 800 | 800 | 98% |
| **Per entire image** | | | |
| *VARIED* | 800 | 500 | 89% |

Figure 11(a). These errors were mostly caused by scenes that contained colour blobs similar to fire. An example of a detected fire per segment in a scene that contains many fire patterns and colours is shown in Figure 11(d). None of the methods recognised this non-fire environment. The biggest problem of per segment approaches is the processing time. To process an image of $1024 \times 768$ pixels, when using per $128 \times 128$ segment approach, it must be divided in a grid-like manner into at least 48 $128 \times 128$ segments which take approximately 5 seconds altogether to process. In the future a heuristic could be employed that would discard regions that definitely do not contain fire which would decrease the processing time.

All these experiments were conducted on a laptop with Nvidia GTX 980 GPU.

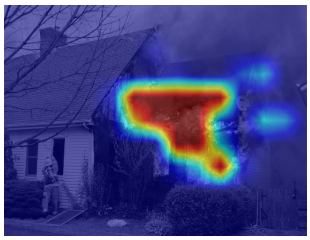
Other examples can be found on the supplementary material website.

(a) Original fire image 1

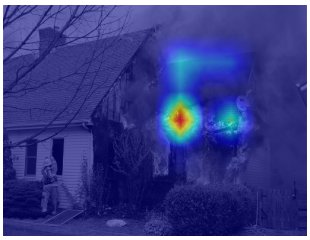
(b) Per segment $128 \times 128$

(c) Per segment $256 \times 256$

(d) Original fire image 2



(e) Per segment $128 \times 128$



(f) Per segment $256 \times 256$

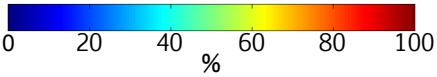


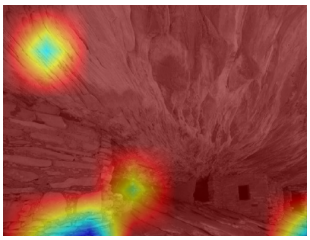

**Figure 10.** Fire detector's results on real fire images of burning houses 1 (a, b, c) and 2 (d, e, f): a, d - original fire images, b, e - heatmap output of the detector trained using $128 \times 128$ pixel samples, c, f - heatmap output of the detector trained using $256 \times 256$ pixel samples, legend: heatmap colours ranging from dark blue (0% probabilty of fire's presence) to dark red (100% probability of fire)

(a) Original non-fire image 1

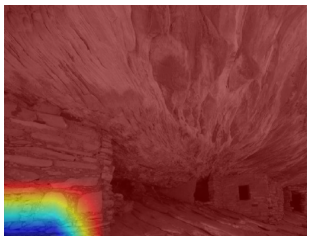
(b) Per segment $128 \times 128$

(c) Per segment $256 \times 256$

(d) Original non-fire image 2



(e) Per segment $128 \times 128$



(f) Per segment $256 \times 256$

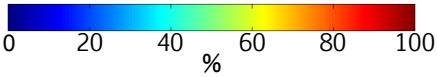
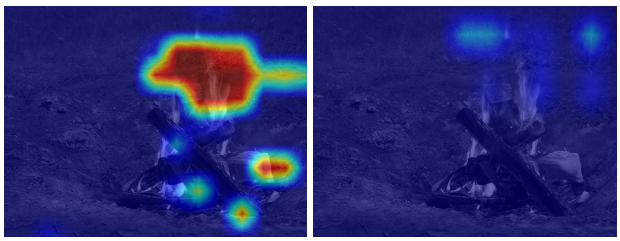



**Figure 11.** Fire detector's results on real non-fire images of a red building (a, b, c) where the $128 \times 128$ segment approach causes a false detection and a cave (d, e, f) that consits of similar patterns and colours of fire which cause many false detections: a, d - original non-fire images, b, e - heatmap output of the detector trained using $128 \times 128$ pixel samples, c, f - heatmap output of the detector trained using $256 \times 256$ pixel samples, legend: heatmap colours ranging from dark blue (0% probabilty of fire's presence) to dark red (100% probability of fire)

## 5. Conclusions

This paper deals with fire detection in image and video using machine learning. It describes a mechanism for artifical fire sequences creation and generation of variable fire samples (shorter animations) from these sequences. These are then used as training data for

(a) Original fire image




(b) Per segment $128 \times 128$ (c) Per segment $256 \times 256$

**Figure 12.** Fire detector's results on a real fire image where the $256 \times 256$ segment approach fails to detect fire: a - original fire image, b - heatmap output of the detector trained using $128 \times 128$ pixel samples, c - heatmap output of the detector trained using $256 \times 256$ pixel samples, legend: heatmap colours ranging from dark blue (0% probabilty of fire's presence) to dark red (100% probability of fire)

a deep convolutional neural network model that forms the base of my fire detector. The detector is implemented using Caffe Deep Learning Framework [11]. Experiments test the detector on an image dataset using two approaches during training — per image and per segment.

I filmed and created 3D models of 2 outdoor scenes. I created 8 different fire simulations in these scenes. Then I rendered 2000 frames of fire animations and composited them with the filmed scenes into complete fire videos. Examples of these frames are shown in Figures 5 and 6. I used them to train 3 different models using 2 approaches to detection. These were tested on similarly modelled fire and non-fire images and the best scored 98% accuracy. For real fire tests, per segment approach with $128 \times 128$ segments reached 50 out of 50 correct fire detections. I reached and surpassed my goal of 95% of true positives detected. Reaching 100% is the most important outcome for a fire detector. False positives reached 24% for this model. Time required to process an image was in units of seconds. $256 \times 256$-segment model reached 5% false positive rate. Improvement might be achieved by training with fire data in more varied environments.

The things and ideas that I consider the most viable are:

- use of a modern method of deep learning for fire detection,
- training of the fire detector based entirely on unreal 3D modelled fire,
- successful experiments on images and an open door for video testing.

In the future work I will add more varied scenes for fire sequences creation. I will continue developing video detection. By adding a temporal element to fire samples, I will see how the detector will be doing and compare these results to its output from still images detection. The sampler and compositer tools may also be useful for other researchers working in this field.

## Acknowledgements

I want to express my gratitude to Doc. Ing. Adam Herout, Ph.D. for his time, materials and professional guidance that he provides me with.

## References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Thou-Ho Chen, Ping-Hsueh Wu, and Yung-Chuen Chiou. An early fire-detection method based on image processing. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 3, pages 1707–1710 Vol. 3, Oct.

[3] J.G. Qunitiere. *Principles of fire behavior*. Career Education Series. Delmar Cengage Learning, 1998.

[4] Turgay Çelik and Hasan Demirel. Fire detection in video sequences using a generic color model. *Fire Safety Journal*, 44(2):147–158, 2009.

[5] Che-Bin Liu and N. Ahuja. Vision based fire detection. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 4, pages 134–137 Vol.4, 2004.

[6] B. Uğur Töreyin, Yiğithan Dedeoğlu, Uğur Güdükbay, and A. Enis Çetin. Computer vision based method for real-time fire and flame detection. *Pattern Recogn. Lett.*, 27(1):49–58, January 2006.

[7] Turgay Çelik, Hasan Demirel, Hüseyin Özkaramanli, and Mustafa Uyguroglu. Fire detection using statistical color model in video sequences. *J. Vis. Comun. Image Represent.*, 18(2):176–185, April 2007.

[8] Hongda Tian, Wanqing Li, Lei Wang, and P. Ogunbona. A novel video-based smoke detection method using image separation. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, pages 532–537, July 2012.

[9] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[10] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[12] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.