NEURAL NETWORK AIDED BURNING RATE DETERMINATION OF ENERGETIC MATERIALS

Jan Langer, Sebastian Knapp, Wolfgang Becker, Andreas Imiolek

Fraunhofer Institut für Chemische Technologie ICT

Joseph-von-Fraunhofer-Straße 7

76327 Pfinztal, Germany

Sebastian.Knapp@ict.fraunhofer.de

Phone: +49 721 4640-643

Abstract

A Convolutional Neural Network (CNN) model was trained using Machine Learning (ML) tools with the aim of automating the analysis of high-speed recordings of solid rocket propellant combustion tests. To obtain sufficient training data for such a CNN, simplified regression images were simulated. Then different model constructions with different input, hidden and output layers were created, trained and evaluated regarding its scoring. The model with the smallest mean square deviation from the expected value was then fed with real data and shows a high accuracy as expected. The model is able to determine different regression lines in different sections with changing slopes, which allows to determine their slopes per section and in the overall average. From this, the burn rate can be calculated automatically with the place and time conditions from the calibration files of the corresponding measurement.

1 Introduction

In the characterisation of new solid rocket propellants, the determination of the burn rate under different environmental influences such as temperature, pressure and gas atmosphere is one of the most important properties for its performance evaluation. To determine the burn rate, the combustion process of the propellant to be characterized is filmed with high-speed cameras and the space and time conditions, i.e. pixel distance and frame rate, are recorded and logged. Subsequently, regression images are generated from these high-speed recordings by adding up the individual pixel values of each frame of a recording into a column vector and concatenating these column vectors of all frames (Figure 1). The combustion front resulting from the vertical burnup can now be recognized in this regression image as a regression line. The burn rate can be calculated from the slope of the regression line together with the recorded location and time conditions. In our case, the determination of the slope of the regression line was previously achieved by manually drawing a line onto the image with the help of small VideoAnalyzer tool. In this work, a possibility is presented to simplify and finally also to automate this complex manual evaluation among other things by the application of a neural network (NN).

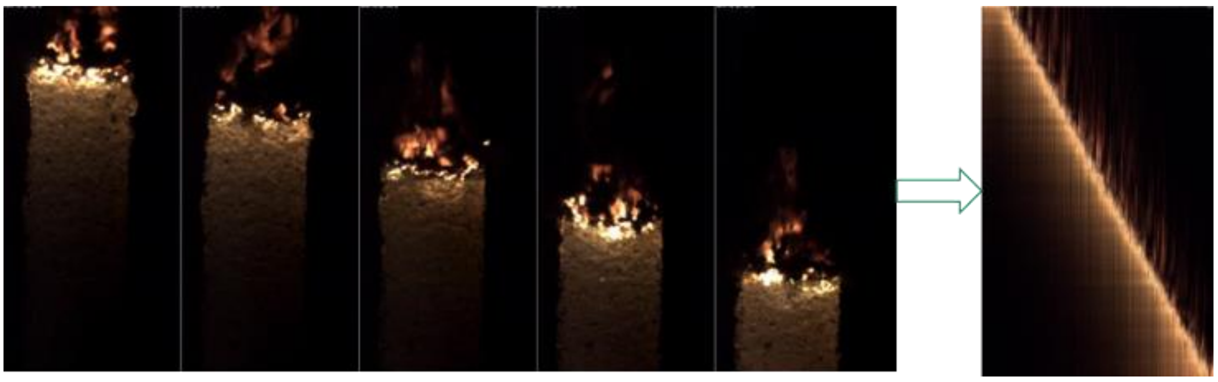

Figure 1: From high-speed recording to regression image

In recent years, the topic of Machine Learning (ML) and Deep Learning (DL) has received a big push, especially because of the increasing availability of hardware that can perform a multitude of parallel operations at affordable prices. A very well-known variant of DL is the NN, which tries to represent structures like decision making and learning processes of a human brain with synapses and neurons by directed and weighted graphs. Nowadays a variety of powerful development tools and frameworks are available, which allow to quickly create small, customized and specialized NNs for different applications without large development teams. The NN here was created in Python using the open source deep learning library Keras in conjunction with Google's TensorFlow backend.

The quality of a NN does not only depend on the structure (the model) but especially on the quality and the amount of available training data from which the model can learn. In order to train the model presented here sufficiently, extra regression images have been artificially created, this will be shown in more detail in Section 2.

The NN presented in this paper is a Convolutional Neural Network (CNN). CNNs are especially suited to evaluate multidimensional input information such as images and are therefore best suited to evaluate the regression images. The mode of operation and the structure of this model are explained in more detail in Section 3.

2 Training

Although a large amount of old data from years of evaluation of combustion experiments would have been available for use as training data, new training data were simulated to train the model presented here, as the compilation, review and processing of the old data would have been too time-consuming. A positive side-effect of using simulated regression images is the resulting objectivity, as this avoids the influence of manual evaluation and thus one part of the human component of the old data. The obvious disadvantage is that the simulated regression images cannot, of course, represent one hundred percent the actual regression images of real experiments, but a well-trained NN, which is not burdened by overfitting, should after all be able to handle new data quite well.

The training of a CNN is a very time-consuming and, due to the multidimensional input data, also a very resource-intensive process, which means that the available hardware is often a limiting factor. In particular, the graphics card(s) is/are decisive here, since graphics cards have far more parallel processing units than CPUs and therefore can train models much faster. Training the model with a CPU would take even longer than with an older GPU. The model presented here was trained with a NVIDIA Quadro M4000, which has 1664 parallel processing units (CUDA cores) and 8GB of graphics memory.

With only 8GB of available graphics memory, the individual training images must not be too large, so that the package size (batch size) of the individual training runs (epochs) does not have to be limited too much, which would significantly affect the scoring of the model after training. After several attempts with different image sizes, some restrictions had to be made regarding format and size of the training images and thus also of the real input images later on.

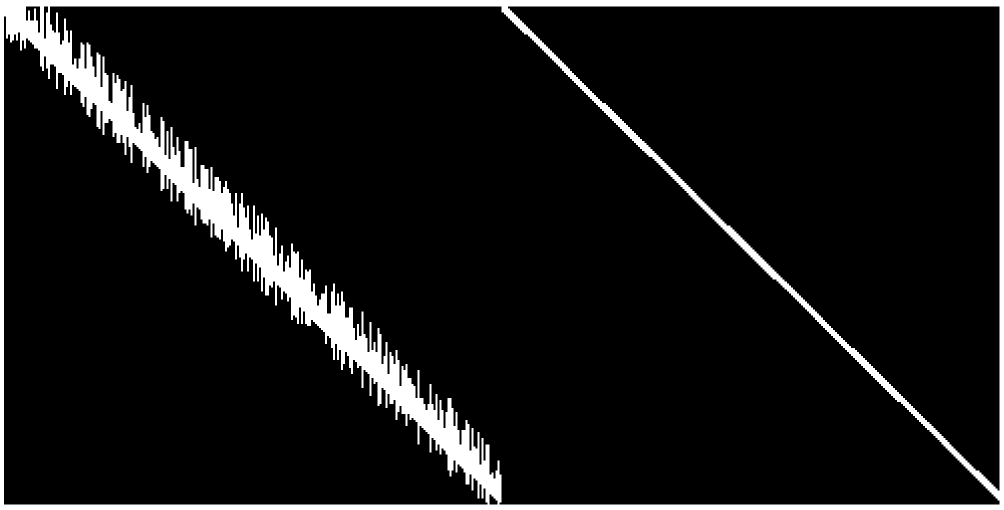

Figure 2: Pair of training images with input image (left) and expected output image (right) with a clear regression line.

Initially only images in grayscale format were used, reducing the colour information dimension from 3 to 1. Furthermore, a format of 248x248 pixels was defined as a fixed size of the input images. As a result, the real regression images must also be converted into a 248x248x1 format before they can be transferred to the model, whereby the aspect ratios must of course be maintained, otherwise the regression line would be distorted. This downscaling means a significant loss of information, especially through the loss of colour information. Thus, depending on the output format, which varies from shot to shot depending on the video length, after scaling down, the images must eventually be enlarged to the square 248x248 size by adding black borders.

The training data consists of image pairs in 248x248x1 format, one input image and one solution/expected output image (Figure 2). Image pairs with different slopes of the regression line and varying degrees of randomly distributed noises along the regression line were generated. In total, nearly 6000 image pairs, i.e. 12000 single images, were generated for training purposes.

With a batch size of 10 image pairs, each epoch of the training took about 70s. As optimization function, the Adaptive Moment Estimation (Adam) function was used and as the loss function, the mean squared error.

After almost 20 epochs a mean square error of less than 0.2% was achieved (Figure 3).

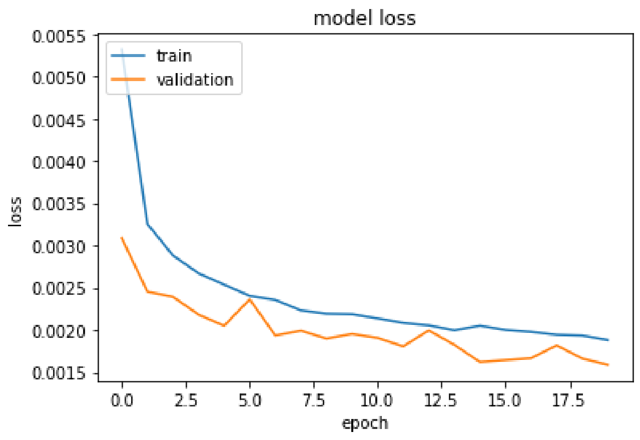

Figure 3: Training log

3 Model

CNNs usually consist of a number of different layers. Besides the obligatory input and output and the name-giving convolution layers, various different filter layers are used. In order to achieve the best possible accuracy of the model, different designs in different variations were trained and tested. The model which finally led to the best result is a sequential model with 31 layers. The basic idea of the final model is to first decompose the input image with the help of the convolution layers, analyse it for its distinctive features and then scale it up again to a suitable regression image. The image is decomposed in a sequence of 7 two-dimensional convolution layers (Conv2D), with a kernel size of 3x3 and a filter size starting with 8 and doubling 3 times up to a size of 64. The size of the input section is steadily reduced due to the filter layers. The activation function for the Conv2D layers is the simple Rectified Linear Unit (ReLU) function. It behaves according to the assignment $f(x) = \max(0, x)$ i.e. it outputs 0 for every negative input, in the other case the value is passed on directly. The additional filter layers which follow up on the Conv2D layers are Dropout and MaxPooling2D layers. The Dropout layers reduce the risk of overfitting during the training process by randomly setting input units to 0 at a rate of 0.2. All inputs that are not set to 0 are upscaled with a value of 1.25 $(1/(1 - rate))$ so that the sum of all inputs remains unchanged. The MaxPooling2D layer filters the input and scales it down by always taking the largest value from a 2x2 pixel window that is moved over the entire two-dimensional input, thus reducing complexity by half. The structure shown in Figure 4: Image decomposition with Conv2D and Max-Pooling layers (simplified) illustrates the decomposition of the input image in simplified form. The last layer of the decomposition has a shape of 27x27x64. Up to this point the model already has 72884 trainable parameters.

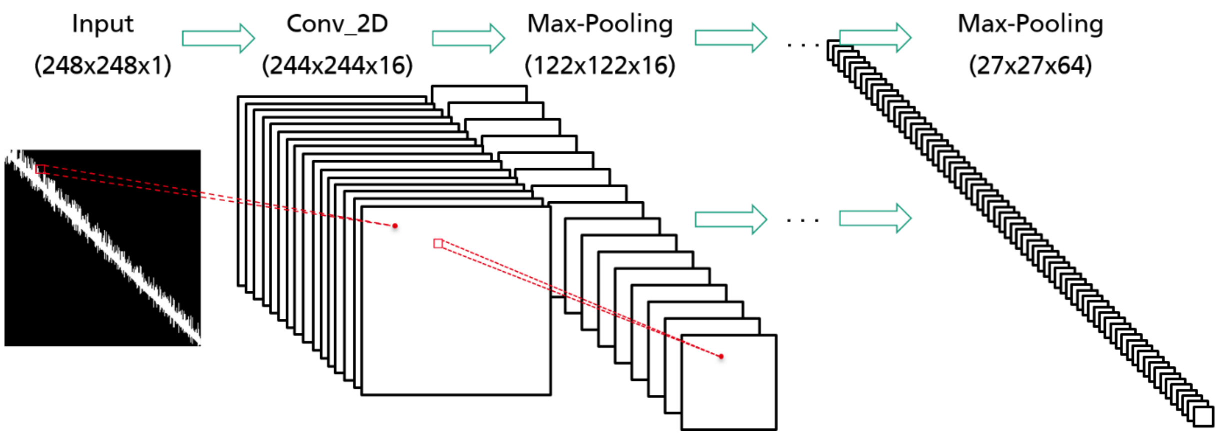

Figure 4: Image decomposition with Conv2D and Max-Pooling layers (simplified)

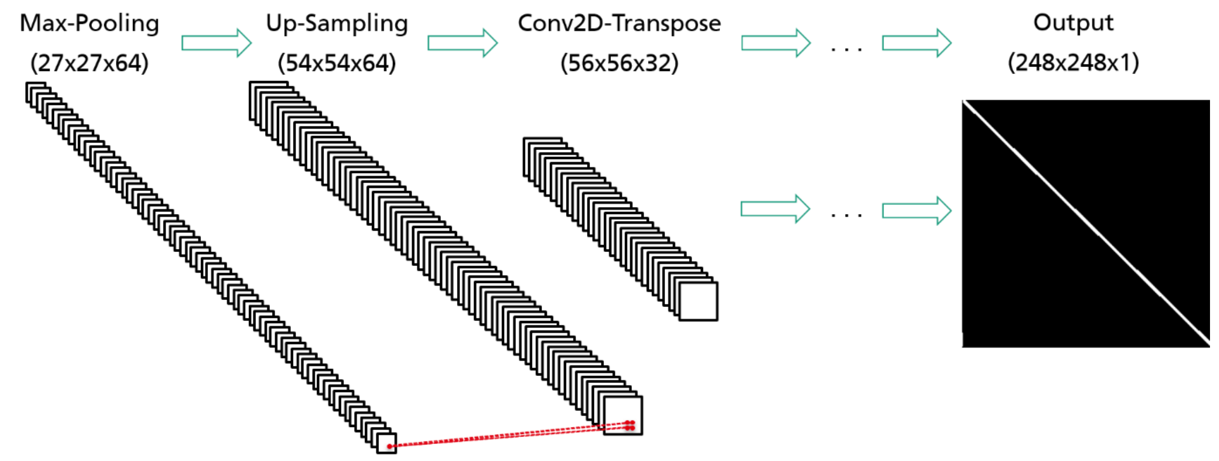

Figure 5: Construction of the expected output image (simplified)

Next, the output image is generated step by step through several layers, as shown in Figure 5. The output image is generated by a total of 10 layers. Up-sampling and transposed convolution layers (Conv2D-Transpose) are used alternately. The up-sampling layers work with a window size of 2x2 and thus quadruple their respective input. The Conv2D-Transpose layers perform inverse convolution operations. They use the same kernel size and filter steps as the Conv2D layers. This results in an output image with the same size as the input image 248x248x1. In this case, the output image in Figure 5 shows the expected output image matching the input image in Figure 4, i.e. a behaviour as it is present during the training. The second half of the model has 108657 trainable parameters, which brings the total number of trainable parameters to 181541.

4 Result

After building the model and training it with our generated training data, it is time to test it on regression images from real measurements. To showcase this, we use the example already presented in Figure 1. This is an HTPB/AP propellant strand that was burnt in a nitrogen atmosphere with a constant flow of approx. 16 kg/h under a pressure of 13 MPa. The video was recorded at a frequency of 150 Hz and the pixel spacing was determined to be approx. 0.015 mm/px using a measurement grid.

Before the resulting regression image can be used as input for the model, it must first be adapted to the input format as explained in Section 2. First, an area of 100 pixels is subtracted from the upper edge to hide the white bar created by the video caption. Then the images are scaled down to a resolution of 248x248, taking into account the original aspect ratio, by filling the missing columns in the x-direction with black pixels. Finally, the colour values of the pixels are scaled to 0 to 1. The adjusted image is then passed as input to the model, which produces the cleaned image shown in Figure 6 as output.

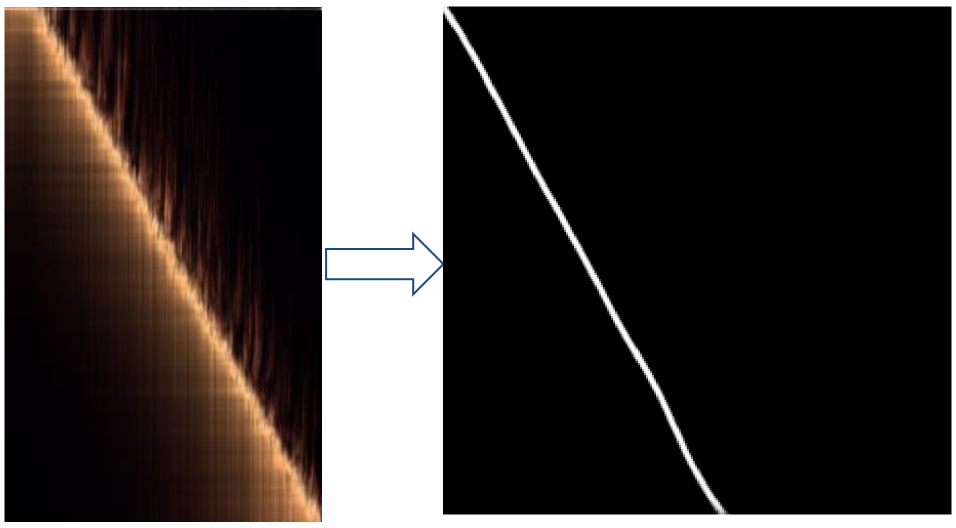

Figure 6: Output of the model

The output image, which has been cleaned of all disturbances, can now be further evaluated with simple image processing tools. This is done here with the help of a small Python script and using functions from the OpenCV2 library. The script determines one or, if necessary, several curve sections in the image if the curve has, for example, jumps or strongly deviating values from a previously determined average slope of a fitted straight line. Thus, a global slope of this straight line or several different local slopes can be determined. In order to be able to draw a comparison to the burn rate previously determined with the original manual method described in Section 1 and since the test in this example had a relatively constant burn rate, the average negative slope of ~1.84 px/px determined by the script is used. The manually determined burn rate is 4.25 mm/s.

With the size and time constrains mentioned above, the burn rate resulting from the model in combination with the evaluation script is:

$$1.84\ \frac{px}{px} \times 150\ \frac{px}{s} \times 0.015\ \frac{mm}{px} = 4.14\ \frac{mm}{s} \qquad \textbf{Eq. 1}$$

The difference between the two results is therefore 0.11 mm/s, which would mean an error of the model of approx. 2.6 % if the 4.25 mm/s were taken as the correct reference. Here, of course, the question arises whether one trusts the value that is generated by a hand-fitted line or the value that is spat out by a "black box" that has been trained via precise training data but has so far had little contact with real data.

5 Conclusion

The question raised in the previous section as to how far one trusts the result of the model presented in this paper cannot really be answered in a qualified manner at this stage, as only 10 real measurements have been evaluated with this model at the time of this paper and although the model presented here has already been revised and optimised several times, there would still be clear possibilities for optimisation. The use of better hardware alone would make it possible to include the colour information from the images instead of just the grayscale format. A huge error factor that has not even been mentioned here, which effects both the manual and the model-based evaluation, is the manual determination of the size constrains by means of the measurement grid. This could be automated in a similar way with image recognition, for example also by means of a CNN.

However, this first attempt to optimise existing evaluation methods with the help of ML tools clearly shows a high potential to simplify these complex and time-consuming tasks. A trained model in combination with the associated scripts for data pre-processing and post-processing could be easily integrated directly into the measurement environments at the test facilities and could provide a precise forecast or even a finished evaluation immediately after a measurement.

Symbols and Abbreviations

API	Application programming interface
CNN	Convolution neural network
CUDA	Compute unified device architecture
DL	Deep-learning
ML	Machine-learning
NN	Neural network
Conv2D	Two-dimensional convolution layer
ReLU	Rectified Linear Unit