
Analysis of the welding process sound using Convolutional Neural Networks for penetration state recognition

Master's Thesis

M. Kolek and S. Zelazny

2020

Aalborg University
The Faculty of Engineering and Science



AALBORG UNIVERSITY
STUDENT REPORT

The Faculty of
Engineering and
Science
Manufacturing
Technology
Niels Jernes Vej 10
DK-9220 Aalborg
www.ses.aau.dk

Title:

Analysis of the welding process sound using Convolutional Neural Networks for penetration state recognition

Theme:

Master's Thesis

Semester:

4th semester of Manufacturing Technology.

Project Period:

Spring semester 2020

Group Members:

Maciej Kolek
Stanislaw Zelazny

Supervisors:

Prof. Simon Bøgh (AAU)

Page numbers:

24 pages

Submitted:

June 19, 2020

Abstract:


With a reliable and accurate method for real-time feedback being a necessity for robotic welding, various methods have over time been proposed for online weld monitoring.

This paper investigates the use of Convolutional Neural Networks (CNNs) to analyze the sound signature from the welding arc, which contains information about the penetration state of the weld pool. The sound is transformed into a spectrogram image using a series of STFT transforms and the image is input into a CNN which classifies it as one of three possible penetration states. Given a pre-recorded dataset, the authors investigate how different sampling strategies impact the ability of the network to generalize to unseen examples. Further it is investigated, how various pre-processing parameters used when creating the spectrogram impact the prediction accuracy of the network. Using the optimal sampling strategy and pre processing parameters a custom CNN is built and tuned, achieving an average testing accuracy of 69,94% in recognizing the penetration state based on a sound sample of 0,25s.

PREFACE

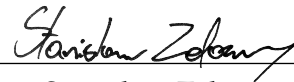
This master's thesis is written by Stanislaw Zelazny and Maciej Kolek during 4th semester of Manufacturing Technology at Aalborg University (AAU). It would not have been able for the hereby presented thesis to be completed without the help of our attentively involved supervisor prof. Simon Bøgh. Our gratitude also goes to Anders Bidstrup and prof. Chen Li for sharing their knowledge regarding the origin of the dataset used and the domain of audio classification. Finally, we must express our very profound gratitude to our parents and to Anička Václavová and Mary Mash for not letting us lose our spirits.

Aalborg University, June 19, 2020



Maciej Kolek

mkolek18@student.aau.dk



Stanislaw Zelazny

szelaz13@student.aau.dk

Reading Guide

The report is written in the style of an academic journal paper, highlighting the points of the work done, its conclusions and considerations. It can be read independently from its appendices, however the curious reader is encouraged to explore the provided additional material, which, besides serving as a proof-of-work, better captures some of the nuances not covered in the main report.

Being a software-based project, the additional material consists of two Jupyter notebooks:

- *Preemphasis filter.ipynb* is an elaboration of how the pre-emphasis filter shown on figure 19 of the report is implemented (link: colab.research.google.com/drive/1C9ck019ECWTDysvJQBajaeEREHD5yaiP?usp=sharing)
- *CNN_factorial.ipynb* Is an elaboration of the 2k-factorial tests of section 4.4 in the report. (link: colab.research.google.com/drive/1mU_W0EnGFUx6LQ1STsE_w698sdVRd73U?usp=sharing)

The notebooks can be opened in Google Collab using the provided links (requires google account). Alternatively static html-versions of the files are available as well, to be opened with a standard web browser. (*CNN_factorial* is then split in two parts, to be read chronologically.)

Additionally, a ROS pipeline for data acquisition has been developed, and is freely available as a git repository at: https://SierraTangoZulu@bitbucket.org/mkolek/welding_setup_packages.git Further materials, as the codebase, dataset and similar are available from the authors upon request.

All units are assumed SI units unless otherwise specified.

Analysis of the welding process sound using Convolutional Neural Networks for penetration state recognition

Maciej K. Kolek¹, Stanislaw Zelazny¹

^a*Institute of Materials and Production, Aalborg University, Fibigerstræde 16, 9220 Aalborg, Denmark*

Abstract

With a reliable and accurate method for real-time feedback being a necessity for robotic welding, various methods have over time been proposed for online weld monitoring.

This paper investigates the use of Convolutional Neural Networks (CNN's) to analyze the sound signature from the welding arc, which contains information about the penetration state of the weld pool. The sound is transformed into a spectrogram image using a series of STFT transforms and the image is input into a CNN which classifies it as one of three possible penetration states. Given a pre-recorded dataset, the authors investigate how different sampling strategies impact the ability of the network to generalize to unseen examples. Further it is investigated, how various pre-processing parameters used when creating the spectrogram impact the prediction accuracy of the network. Using the optimal sampling strategy and pre-processing parameters a custom CNN is built and tuned, achieving an average testing accuracy of 69,94% in recognizing the penetration state based on a sound sample of 0,25s.

Keywords:

MIG-Welding, Welding sound analysis, Convolutional Neural Networks.

1. Introduction

Since it is industrial breakthrough in the 1940s electric arc welding has been widely adopted as a major joining method in a wide variety of industries ranging from consumer products, to automotive, aerospace and construction. As welds are often used in critical construction elements a wide body of research, dedicated to the task of weld quality inspection, has steadily evolved alongside new welding technologies. (Cary, 2004)

With the growing adoption of robotic welding, its focus has in the recent two decades increased towards methods of on-line welding inspection, enabling real time feedback control of the process making it more robust and preemptively reducing the amount of welding defects. Different approaches such as spectroscopy (Zhang & Chen, 2014) (Zhang et al., 2013) and visual sensing (You et al., 2014), radiography (Guo & Rokhlin, 1990), video (Zhang et al., 2019), (Liu et al., 2017) or through the arc sensing have been proposed. However, the high temperature, electric current and brightness of the welding arc has always posed an additional challenge for sensor protection and sensor placement.

1.1. Use of sound for weld monitoring - State of the art

Using the sound from the welding arc instead has already been proposed as early as in 1969 (Jolly, 1969), and in 1998 Saini & Floyd (Saini & Floyd, 1998) made the first mentions of using it

as a data source for online robotic feedback. A big advantage of this mode of sensing is the ease of data acquisition requiring nothing more than a microphone within the human auditory range (20Hz to 20kHz). Any reader familiar with welding will know, that the auditory feedback is an important cue for the manual welder in controlling the welding process, as formally proven in experiments by (Tarn & Huissoon, 2005).

Apart from this human intuition, numerous studies have confirmed that the acoustic signature of the welding arc contains rich information about the state of the process: Rostek (Rostek, 1990) already in 1990 discovered that a wide variety of process parameters, including voltage supply, wire feed rate and even shielding gas flow could be traced back from the arc sound.

Horvat et al. (Horvat et al., 2011), (M Čludina, 2003), (M. Cudina, 2008) were able to, using a physics-based model for the relation between the welding arc and generated sound pressure, with high accuracy reconstruct the welding sound from the measured arc current. Chen et al. (Lv et al., 2014) proved that the arc length can be predicted using a wavelet-packet decomposition based noise filter, even enabling a feedback loop for arc length control. Kamal Pal et al. (Pal et al., 2009) were able to correlate the weld transfer mode and deposition efficiency (percentage of useful electrode material in the weld) to statistical values in the time domain of the sound signal.

As, according to Horvat et. al, (Horvat et al., 2011) the sound is related to the energy of the arc and the sound wave is actively affecting the molten welding pool, there have been numerous studies for using the welding sound for predicting the weld penetration depth or penetration state (e.g. lack of penetration or weld burn trough) (see Table 1 for references).

1.2. Machine Learning as an analysis tool

With the different in- and output parameters of the welding process being present in the welding sound, as listed in the previous section, numerous methods for extracting that information have been proposed. Early work such as Saini & Floyd (Saini & Floyd, 1998) or Kamal Pal (Pal et al., 2009) (Pal, 2011), selected a statistical property of the sound (e.g. signal kurtosis) and investigated its correlation with a certain weld property, (e.g. penetration depth,) but with the new era of Machine Learning (ML) the trend has shifted towards extracting multiple features from the signal, and then applying ML algorithms for finding hidden correlations within the data.

Table 1 summarizes a review of publications on the topic over the past decade. Most listed methods rely on first dividing the recording into smaller samples, (everywhere from sub-second to 4 seconds length) and then extracting different features from the temporal (e.g. total signal energy, rms, kurtosis) and frequency domain (e.g. spectral centroid, frequency rms, MFCC-coefficients etc.) The features are then collected in a vector and fed into the selected algorithm. Different variations of Neural Networks are used: (Saad et al., 2006) uses a traditional Artificial Neural Network (ANN) with back propagation, but newer ML flavors are in use as well: (Wu et al., 2017) uses a Deep-Belief Network (DBN), fed with a combination of video and audio features, while (Lv et al., 2016) uses the Adaboost method combine multiple ANNs into one strong classifier.

As training a network on a full set of features is more time consuming, in order to improve performance (as real-time feedback control is often the end-goal) different methods have been used to reduce the feature space and only extract the features which prove to be relevant. Notable mentions include (Wang et al., 2011) using Genetic Algorithms (GA) to reduce the feature length from 128 to 12, or (Wu et al., 2017) using t-stochastic neighborhood embedding (t-SNE) to reduce the initial high-dimensional feature set to a low dimensional eigenvector input.

A whole different approach is taken by (Ren et al., 2018) who use a Convolutional Neural Network (CNN), a neural network architecture commonly used in the field of image classification.

Instead of a vector of pre-extracted features, their CNN is fed with an 129x32 time-frequency image of the sound sample, (i.e. a spectrogram), and the features are learned directly by the network from the input. The main advantage of this is making it more robust to the designers choice of features and noise in the data. When compared by the researchers to a traditional ANNs the same dataset, the CNNs also outperformed the ANNs on testing accuracy and stability of results.

1.3. Project background and Motivation

The project described in this paper stems from original work on Aalborg University by Anders Bidstrup on using ANNs for recognizing penetration state and transfer mode in DC MIG welding (Bidstrup, 2017). While CNNs have been in existence for long, and are widely used in the field of image analysis, there are very few examples of this powerful method being used on industrial non-image data, especially in the field of welding inspection. Continuing on Bidstrup's line of research, this project investigates and applies the CNN architecture to his experimental results, in an attempt to further analyze the differences between the two methods as well as to characterize what parameters influence the performance of CNNs in this uncommon application.

Section 2 introduces the dataset used for the project and how the data was acquired. Sections 2.2 and 2.3 give an initial analysis of the dataset and investigate the effect of welding settings on the acquired data. Before being able to design the main architecture of the network, sections 3 and 4 investigate the choice of an adequate sampling strategy for training and how the preprocessing parameters affect the image input. After choosing the input parameters the main CNN architecture is designed and tuned in Section 5. Finally, the results are reviewed and concluded in Section 7, with recommendations for future work.

2. Dataset

Due to the Covid-19 pandemic it was unfortunately not possible to conduct laboratory experiments and gather new data for the research project. The work was therefore continued on the original dataset from (Bidstrup, 2017) providing an opportunity to compare the two methods directly. (A software pipeline for easy data acquisition based on the ROS robotic platform was also developed, see details in accompanying material.)

2.1. Data collection

The welding sound dataset was originally acquired by Bidstrup in 2018 using the welding setup shown on Figure 1. It consists of an ABB IRB 140 robotic arm, equipped with a welding gun and a omnidirectional microphone mounted at a low angle approximately 30 cm from the welding pool. The welding current and voltage were also measured and collected alongside with the sound.

The welding sample consisted of two 3mm mild steel plates of length 150 mm, spaced 2mm apart in order to be joined with a square butt weld. The essential parameters of the setup are listed in Table 2, with a more detailed description of the setup available in the original document.

The main goal of the experiments was to capture sound from 3 different penetration states: lack of penetration, normal penetration and excessive penetration, illustrated on Figure 2, encoded as c_1 , c_2 and c_3 . (A smaller dataset containing different metal transfer modes was captured, but will not be used in this work.)

| Reference | welding process | input features | algorithm | output |
|---------------------------|-----------------------------|-------------------------------------|------------------------|-----------------------|
| (Pal, 2011) | P-MIG [] | sound curtosis | descriptive statistics | Penetration depth |
| (Pal et al., 2009) | P-MIG [] | sound curtosis | descriptive statistics | Deposition efficiency |
| (Sipei Zhao & Lele, 2018) | (P) ¹ -GMAW [DC] | 10 td, 4 fd | GMM | Transfer mode |
| (Ren et al., 2018) | P-GTAW [AC] | fd: (129x32) spectrogram image | CNN | Penetration state |
| (Lv et al., 2016) | P-GTAW [AC] | 23 td/fd | Adaboost- ANN | Penetration state |
| (Dong et al., 2017) | P-GTAW [AC] | fd: 1501 STFT frequency bins | PCA | Penetration state |
| (Wu et al., 2017) | VPPAW ² | 2 vi ³ , 5 td, 12 fd | t-SNE, DBN | Penetration state |
| (Saad et al., 2006) | VPPAW | 9 fd (Welch PSD) | ANN (1 layer) | Penetration state |
| (Wang et al., 2011) | GTAW [] | 12 td | GA, ANN (1 layer) | Penetration state |
| (Bidstrup, 2017) | MIG [DC] | 18 td, 32 fd, 1134 wpd ⁴ | ANN (2 layers) | Penetration state |

P = Pulsed [..] = Welding current (AC/DC), if specified td = time domain features fd = frequency domain fetures

¹ - Both pulsed and continuous welding experiments conducted.

² - VPPAW: Variable Polarity Plasma Arc Welding [AC].

³ - Visual. Two features describing the pool shape were extracted from a video feed.

⁴ - Wavelet Packet Decomposition. A WPD tree of depth 5 is used, at each node, 18 features are extracted, summing up to $18 \cdot (2 \cdot 2^5 - 1) = 1134$ See Appendix Appendix A for details.

Table 1: Overview of reviewed recent literature, summarizing different approaches to predicting process outputs from the welding sound.

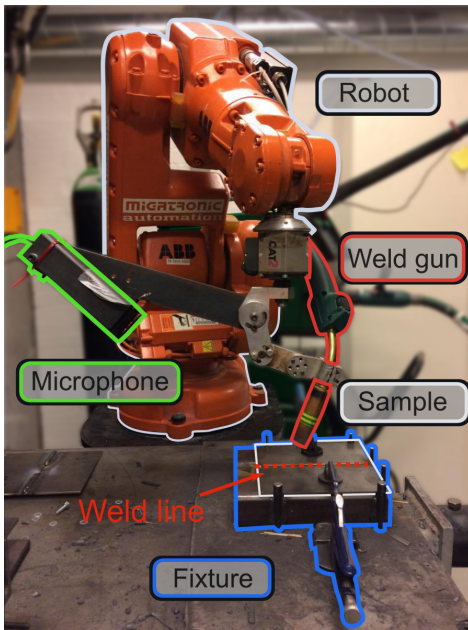


Figure 1: Welding setup overview (Bidstrup, 2017) (modified)

A total 53 welds were performed with varying welding settings, summarized in Table 3. First, two runs of 10 welds each were performed in order to collect sound samples of all the penetration states. Due to a lack of enough data representing lack of penetration, a series of further 14 runs were performed, varying the welding settings as shown on the table, in order to provoke it. After the experiments, the welded samples were investigated and segments of the welds were categorized according to the guidelines in ISO-5817. The corresponding sound recordings were then likewise cut up into files and labelled, resulting in a dataset of 77 files.

| Equipment | |
|-----------------------|---------------------------------------|
| Robot | ABB IRB 140 |
| Welding equipment | Migatronix Flex 400 |
| Microphone | Projects Unlimited AOM-673P-R |
| Data acquisition card | NI USB-6216 M |
| Sound Card | Roland Edirol UA-25 |
| Weld sample | |
| Material | S235JR |
| Root gap | 2 mm |
| Thickness | 3 mm |
| Welding settings | |
| Gas | Mison 18 (18% CO ₂ 82% Ag) |
| Gas flow rate | 14 l/min |
| wire diameter | 1,2 mm |
| Travel speed | 2 mm/s |
| work / travel angle | 82°/22° |
| CTWD | 5 mm |

Table 2: Summary of equipment and welding settings used for data acquisition.

2.2. Preliminary data analysis

The original experiments yielded a total of 3091 seconds (51m 31s) of sound. Due to uncertainties some of the files were discarded by the author, resulting in a final pruned dataset of 2856 seconds (47m 37s) and 66 files. A preliminary analysis of the dataset revealed some possible challenges lying ahead:

- While some welding settings shown on Table 3 were able to produce welds with 100% normal or excessive penetration, it has not been possible for Bidstrup to consistently produce lack of penetration, resulting in multiple penetration states occurring in some of the weld samples. This casts doubt to whether there might have been ambiguity in classifying the fragments of the weld bead, as well as indicates that the welding process is unstable and

| Voltage 22V+n | WFS [mm/s] | Rep. | Exp. | c ₁ | c ₂ | c ₃ |
|------------------|---------------|------|---------------------------------|----------------|----------------|----------------|
| -2,8 | 5,5 | 10 | c ₂ | 17% | 83% | 0% |
| -3,5 | 7 | 10 | c ₁ , c ₃ | 11% | 3% | 86% |
| -1,5 | 7 | 4 | c ₁ | 8% | 0% | 92% |
| -2,5 | 7 | 1 | c ₁ | 0% | 0% | 100% |
| -4,5 | 7 | 1 | c ₁ | 0% | 0% | 100% |
| -2,5 | 4,5 | 1 | c ₁ | 0% | 100% | 0% |
| -2,8 | 4,5 | 2 | c ₁ | 29% | 71% | 0% |
| -3,0 | 4,5 | 2 | c ₁ | 50% | 50% | 0% |
| -3,0 | 5 | 1 | c ₁ | 20% | 80% | 0% |
| -3,0 | 6 | 1 | c ₁ | 0% | 100% | 0% |
| -3,0 | 7,5 | 5 | c ₁ | 24% | 0% | 76% |
| -3,2 | 5,5 | 5 | c ₁ | 29% | 71% | 0% |
| -3,3 | 5,5 | 1 | c ₁ | 36% | 64% | 0% |
| -3,5 | 5 | 4 | c ₁ | 25% | 75% | 0% |
| -3,5 | 6 | 4 | c ₁ | 13% | 12% | 75% |
| -3,5 | 6,5 | 1 | c ₁ | 13% | 9% | 78% |

WFS = Wire Feed Speed Rep. = repetitions Exp. = Expected state

Table 3: Overview welds performed to obtain dataset. The percentages represent the actual distribution of the penetration states in the samples of each run (i.e. one weld can contain multiple states). The voltages represent adjustments from a base voltage of 22V.

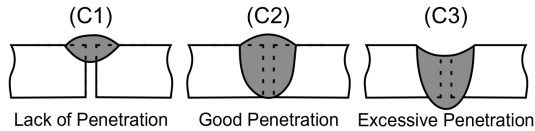


Figure 2: Illustration of the three penetration states, encoded as c_1 , c_2 and c_3 .

not in full control, as illustrated on Figure 3. This also means, that as the welding process was crossing the different penetration regimes, some of the labelled sound data represents a transition between the categories, instead of representing a "pure" (i.e. stable) penetration state.

- The dataset has an unequal distribution of the 3 classes, with c_1 , lack of penetration being underrepresented in the dataset, as shown on figure 4. The files for that class are also on average shorter. This requires extra steps in order to equally present the 3 classes to the training algorithm, as further described in section 3.
- The different penetration states were induced by changing the welding settings. This essentially changes multiple factors at once between the experiments, making it harder to separate changes in the sound attributed to the penetration state from changes resulting from weld settings. Only three runs of experiments (run 2, 15 and 16) contain all 3 classes while providing constant weld settings, however each of them contains at most only 20s of

sound from the least represented class, which is insufficient for a training set.

- As described by the author, the data was collected over several days, with different background noise conditions. However the dataset contains sound clips with the silence before and after the weld already removed. While background noise (mainly ventilation) is evidently audible in the data, there is no reference data of the pure ambient noise available for comparison. This leads to speculation that the harmonic artifacts (horizontal stripes) seen in some of the spectrograms, eg. those shown on Figure 16 may originate from the ventilation system.

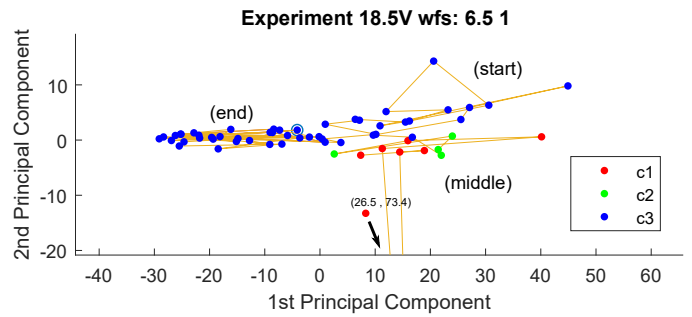


Figure 3: Principal components for the sound features of one whole welding experiment (60s), plotted second by second. The changing penetration state and erratic behaviour of the sound, suggest that the welding process is not stable.

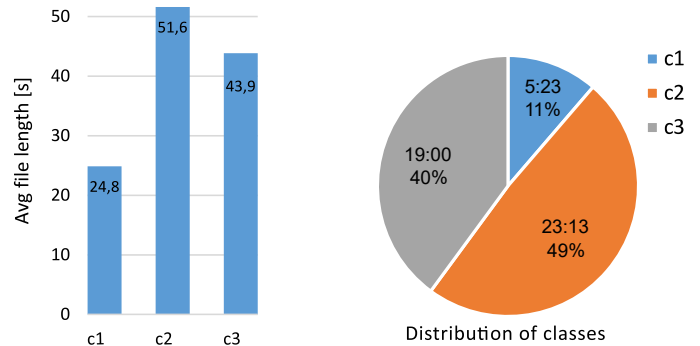


Figure 4: Average file length and percent-wise data distribution by class. (One full weld contains approx 64s of sound)

2.3. Effect of welding parameters

As listed in the previous subsection, there was concern whether changing the welding parameters (voltage and WFS) in between experiments would alter the behavior of the sound, posing an additional source of variance in the dataset. Bidstrup's original ANN extracted over 1000 various time and frequency domain features from the sound samples (an overview is available in Appendix A). Using his choice of features as a point of origin, the same set of features were extracted for each second of the sound data, then concatenated and averaged for each file. A

Principal Component Analysis (PCA) was performed on the resulting set of file-wise feature vectors and plotted as shown in Figure 5. The figure shows, that while the three classes can in a general sense be clustered together (c_1 is generally on the left side of the chart, c_3 occupies the middle and c_2 the right), the welding voltage (marker color) also consort the files into groups. (A similar, albeit less consistent, pattern has also been noted if grouping the experiments by WFS.)

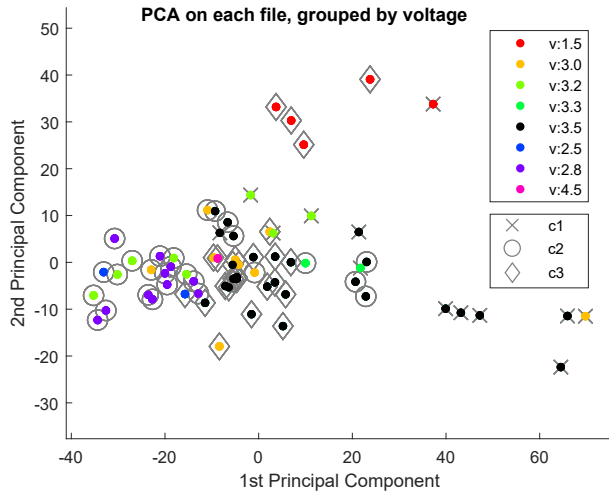


Figure 5: Principal Components of the average sound features for each whole file in the dataset.

It was therefore chosen to select the -3.5V group (marked black on Figure 5) containing all 3 classes as a reference subdataset: v_{35} which can be used for comparing the CNN performance on data with and without varying weld parameters.

It is worth noting, that each point on Figure 5 represents a whole file, which spanning over several seconds can contain much more variance within. Figure 6 shows therefore the PCA components of each second of sound individually. As it can be seen from the figure, the three classes are highly overlapping, making it difficult to assign an unlabelled point to a cluster based on the main PCA values alone, (the first 3 Principal components only explain 45% of the dataset variance.)

Besides motivating the use of a more advanced method, (like the CNN,) plotting the second-wise PCA components for the v_{35} set, shown on figure 7, shows an almost identical location of the 3 clusters. When grouping the points by filename, as shown on for Figure 8, the different files overlap in a 'confetti' pattern, indicating that there is no significant difference between the individual files. This means the following can be concluded about the v_{35} and main dataset:

- The PCA clusters for the three classes are highly overlapping as illustrated on Figure 6 and the two subplots in Figure 7, indicating that the welding sound of the three classes is very similar, (i.e. the features identifying the class are not among the main sources of variation in the dataset,) which is motivating the use of a more sophisticated classification method.

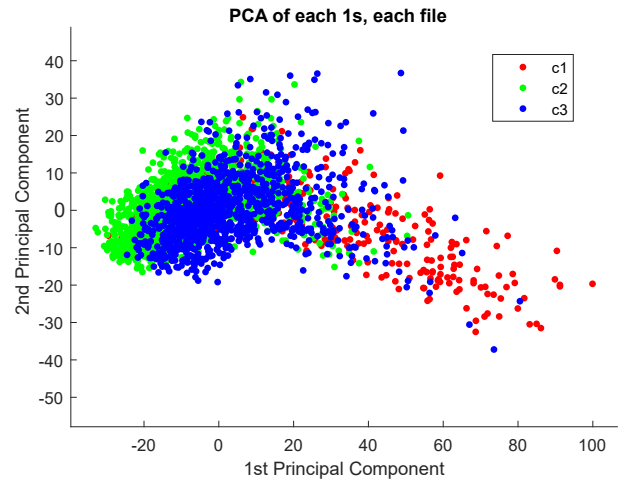


Figure 6: Principal Components for each second in the dataset.

- The shape and location of the clusters in the main dataset, shown on Figure 6 and the v_{35} set, shown in Figure 7 is very similar, meaning there is no apparent difference in what features are identifying a certain class. (i.e. training done on the 'cleaner' V_{35} may be transferable.)
- As shown on Figure 8 the groups from each file within a class in v_{35} overlap, (with exception of the single file highlighted in Figure 7, removed from the set) meaning that, despite having different WFS the files within the v_{35} set are similar, requiring no further subdivisions.

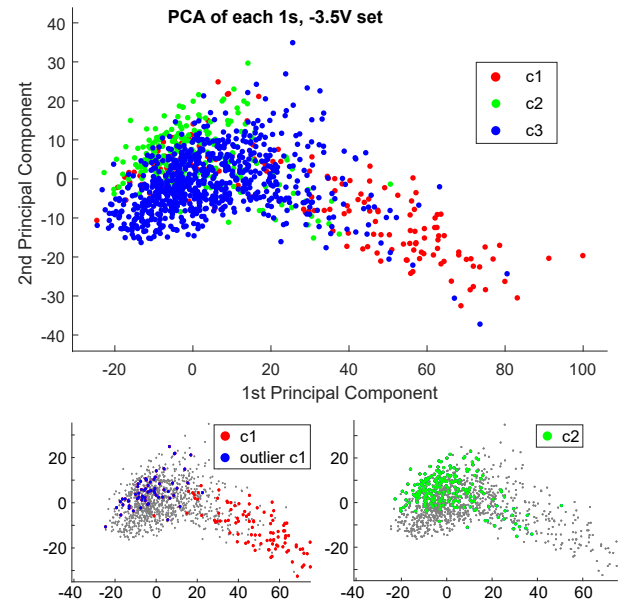


Figure 7: Principal Components for the sound features of each second of sound in the v_{35} set. Figure on lower left highlights one c_1 file in the set, significantly different from its peers, a possible mislabeling.

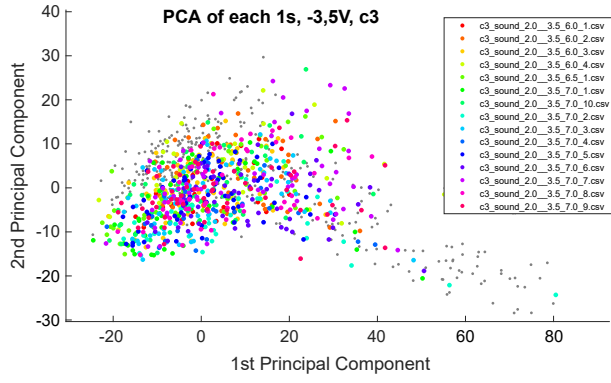


Figure 8: Principal Components for the sound features of each second of sound in the v35 set, c_3 files highlighted and grouped by filename.

Further figures illustrating the above mentioned conclusions from the analysis can be found in Appendix B.

3. Sampling strategy

Before a CNN can be trained an important consideration to make, is how to manage the gathered data in order to avoid overfitting and design a testing scenario that would truthfully represent the anticipated performance in a real use case. In order to compensate the uneven distribution of the classes and the lack of data from the same set of welding parameters, discussed earlier in Section 2.2, the training samples are generated from the dataset as follows:

1. First, the last 20% of each file is truncated and stored in a separate testing set, as illustrated in Figure 9, and won't be seen by the algorithm during training and validation, .

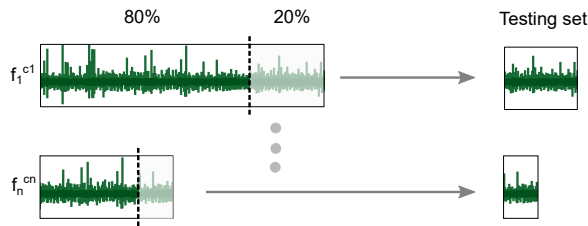


Figure 9: The files being truncated to generate a training/validation and testing set.

2. The remaining files are then sampled in order to create spectrogram images for training the CNN. Generally, more images are needed than there is seconds of welding sound, meaning that some data is sampled more than once. As the three classes are unevenly represented in the dataset (see Figure 4), the number of samples, n_s is calculated by first deciding on how much class c_1 will be "oversampled" i.e. how many times the length of all c_1 spectrograms will exceed the length of the c_1 sound data:

$$n_s^{c_1} = \sum_{n=1}^{n^{c_1}} \text{len}(f_n^{c_1}) \cdot \frac{1}{l_s} \cdot n_{\text{times}} \quad (1)$$

Where n_{times} is the amount of oversampling, $\text{len}(f_n^{c_1})$ is the length of the n -th c_1 file in the dataset, l_s is the length of one sample and $n_s^{c_1}$ is the resulting number of c_1 samples. Once the number is set, the total number of samples is set as $N_s = 3 \cdot n_s^{c_1}$. The N_s samples are then drawn one by one, with each class having an equal probability of being selected, ensuring all three classes are represented equally in the sample pool.

Once a class is selected to be drawn from, the sample is taken by selecting a random starting point in a random file in the given class, with the probability of a file being selected proportional to the file length, as illustrated in Figure 10. This ensures that the average overlap of sample windows is the same for each file within a class, and the random location of the starting point means that even though $n_{\text{times}} > 1$, no samples are identical.

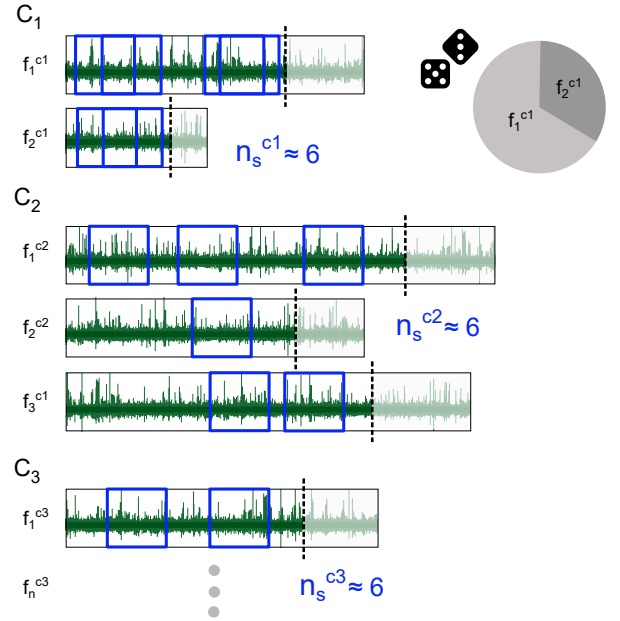


Figure 10: The files being sampled. Each class has an equal probability of being selected, resulting in an approximately equal $n_s^{c_n}$. The probability of a file being selected is proportional to file length within its class, giving an equal density of samples for all files in a class.

Expectably, if two sampling windows overlap, the same features of the sound will be present in both resulting spectrograms, although the location of the feature will be shifted, as illustrated in Figure 11. This bears similarity to the practice of augmenting visual image data by rotating or cropping the images, in order to enlarge the dataset, as well as make the algorithm more robust against overfitting (Géron, 2019).

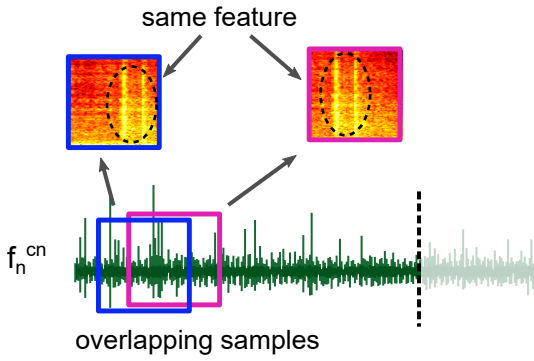


Figure 11: Overlapping sample windows, resulting in the same feature being present in both spectrograms.

3. After the samples are selected, they are transformed into spectrogram images, as described in further detail in Section 4. The images are then randomly split into 80% used for training and 20% for validation, as shown in Figure 12.

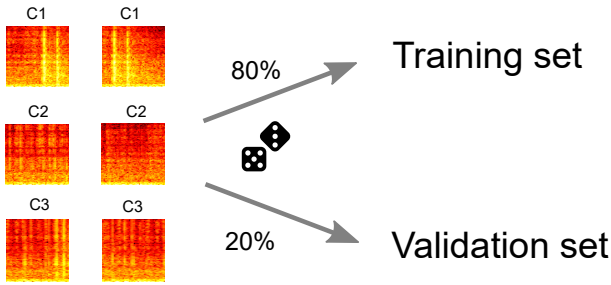


Figure 12: The pool of samples is randomly split into 20% for validation and 80% for testing.

4. After the CNN is trained its performance is tested using the test data truncated in step 1. Each file in the test set is traversed sequentially with no overlap, each sample being classified by the neural net, as illustrated in Figure 13. This mimics data incoming sequentially from a welding setup and being classified in real-time for on-line weld monitoring. The predictions are then compared to the actual file labels, and an overall average accuracy acc_u is calculated as an expression of the networks performance:

$$acc_u = \left(\frac{n_{corr}^{c_1}}{n_{ts}^{c_1}} + \frac{n_{corr}^{c_2}}{n_{ts}^{c_2}} + \frac{n_{corr}^{c_3}}{n_{ts}^{c_3}} \right) \cdot \frac{1}{3} \quad (2)$$

Where n_{corr} is the number of test samples predicted correctly and n_{ts} is the total number of test samples for a given class. It is worth noting, that acc_u is not weighted with respect to the relative size of the n_{ts} , which puts more emphasis on recognizing all three classes equally, instead of achieving high accuracy on classes representing a majority of the dataset.

The effect of choosing different numbers of samples was investigated using an initial CNN architecture described in Section 5. Holding the parameters for generating the spectrogram

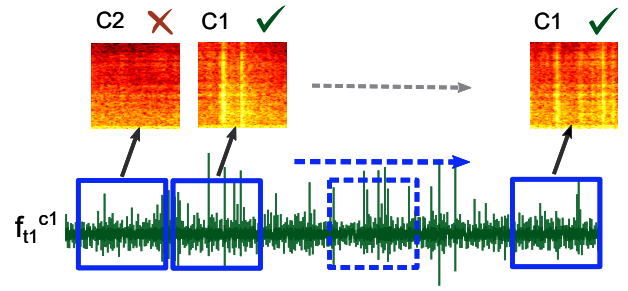


Figure 13: Testing: the sound files from the test set are scanned sequentially, with each sample window labeled by the algorithm. The predicted window labels are then compared with the actual file class and the accuracy is calculated.

Sampling density test:

| | |
|-------------------------|-------------|
| step length l_{step} | 0,5 / 0,25s |
| window length l_{win} | 256 |
| overlap | 50% |
| pre-emphasis | 0,0 |
| epochs trained | 20 |

Table 4: Pre-processing parameters for the tests shown in Figure 14.

constant, (see Table 4) the neural net was trained with samples generated by different magnitudes of n_{times} , with the results as shown in Figure 14. As seen from the upper graph, even a ten-fold oversampling does not induce overfitting: i.e. although the training samples are highly similar due to the amount of overlap the algorithm is still able to generalize the information to the unseen test set.

Although the overlap is always largest for the samples in the c_1 class, the amount oversampling does not seem to show any trend on the prediction accuracy of the individual classes, as shown on the lowest graph. Comparing the upper and middle graph of Figure 14 shows, that the relative performance of different sample lengths is better assessed when compared at the same n_{times} value than the absolute number of samples, as it directly expresses how many times the information in the data set has been seen by the algorithm,

In general, acc_u increases linearly with n_{times} , however the larger number of samples also increases the pre-processing time for generating the spectrograms. As a trade-off it is decided to continue all further experiments with an $n_{times} = 2$, when investigating the pre-processing parameters in the next section.

4. Pre-processing

Before the data from the sound recording can be fed into the neural network, it first needs to be converted into an image representation that the CNN architecture operates on. This section investigates the process of converting the time series data from the sound into a spectrogram and the influence of different pre-processing parameters on how the information contained in the sound is exposed for the CNN to train on.

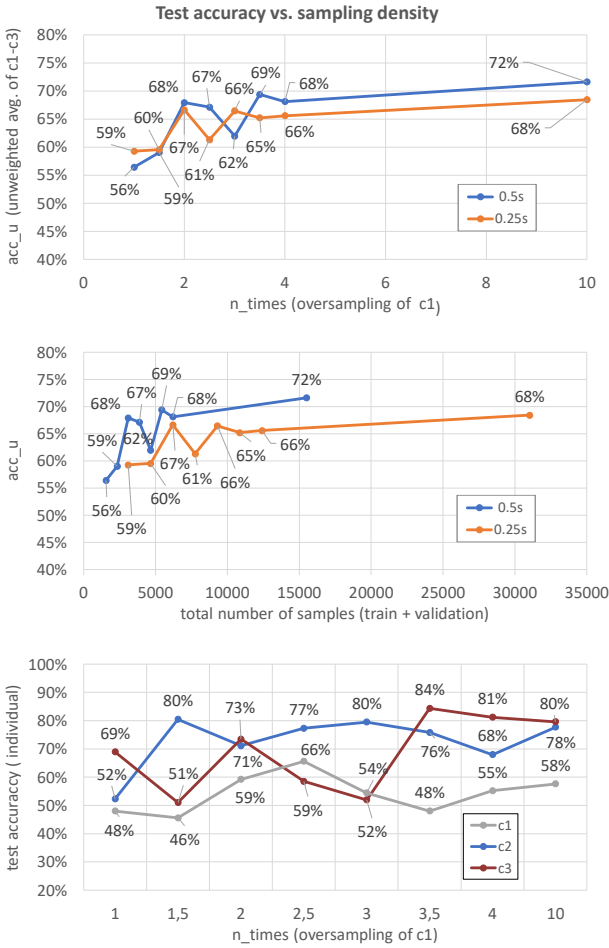


Figure 14: The overall accuracy acc_u and accuracy of individual classes under different n_{times} or number of samples. The lowest graph is for sample length of 0.5s.

4.1. Spectrograms

The spectrogram represents the intensity of various frequencies present in the time series, shown over time. The pipeline for creating a spectrogram is illustrated in Figure 15 and consists of four main steps:

1. First, as discussed in section 3, a sample window of a certain length, denominated as the *step length* (l_{step}), is taken out of the signal.
2. The sample is then again windowed using a sliding windowing function (here, a Hamming window) of a given *window length* (l_{win}), which is moved by a certain *hop length* (l_{hop}). The hop length is typically calculated from a specified *overlap* in percent, which expresses the fraction of the windowing functions overlapping:

$$l_{hop} = l_{win} \cdot (1 - overlap) \quad (3)$$

Because of its connection to the STFT in the next step, l_{win} and consequently also l_{hop} are expressed in audio samples and always chosen as a power of two, while l_{step} is expressed in seconds.

3. Each of the windowed bits of data is then passed through a Short Time Fourier Transform (STFT), which calculates the intensity (converted to dB) of the various frequencies present in the signal. The intensities are returned as a vector of $l_{win}/2 + 1$ frequency bins.
4. The vectors from each STFT are then concatenated column-wise into a spectrogram image of the whole sample. Each column of pixels represents the frequency spectrum of the sound at a given point in time, with the rows of the image representing the frequency bins. The dimensions of the whole image therefore depend on the parameters chosen and are calculated as follows:

$$height = \frac{l_{win}}{2} + 1 \quad width = \frac{f_s \cdot step}{l_{hop}} \quad (4)$$

Where f_s is the samplerate of the audio signal.

5. Once all of the N_s images are created, the whole dataset is normalized, mapping all pixel values from 1 to 0 (i.e. the brightest pixel in the *dataset* is mapped to 1, preserving the relative "loudness" difference between spectrograms.)

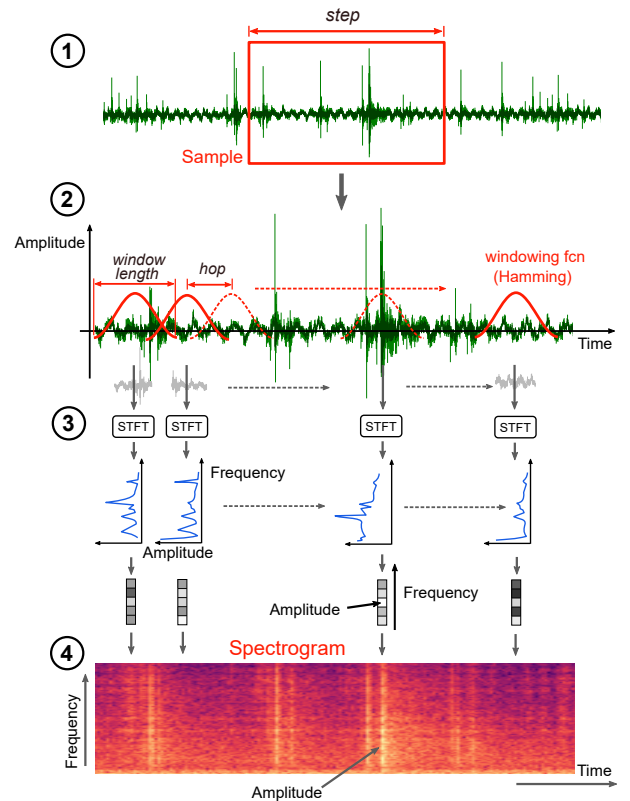


Figure 15: Conversion of time series data into a spectrogram image: 1) the sample of data is selected, 2) the sample is windowed using a sliding windowing function, 3) The windowed data is passed through a Short Time Fourier Transform (STFT) outputting a vector of amplitude intensities, 4) The vectors are concatenated in the time dimension to form a spectrogram image.

4.2. Pre-processing parameters

Besides the image dimensions, the parameters in equations 3 and 4 also affect how the information carried in the sound signal is accentuated. The correct choice of those pre-processing parameters is therefore crucial for achieving good recognition performance. This section investigates the meaning of each of the parameters and how it influences the appearance of the spectrogram image.

The first parameter to consider is l_{step} i.e. how long a segment of the sound a single image represents. Unlike other authors like (Ren et al., 2018) or (Wu et al., 2017) working with AC welding, where the optimal step size can easily be chosen from the frequency of the welding current, in DC welding the electrical arc, the acoustic impulses from the ignition of the electrical arc¹ are irregularly spaced apart as visible in Figure 16. If the pattern of the impulses is in any way correlated to the penetration state of the weld, it is important to select a step size long enough for it to be included.

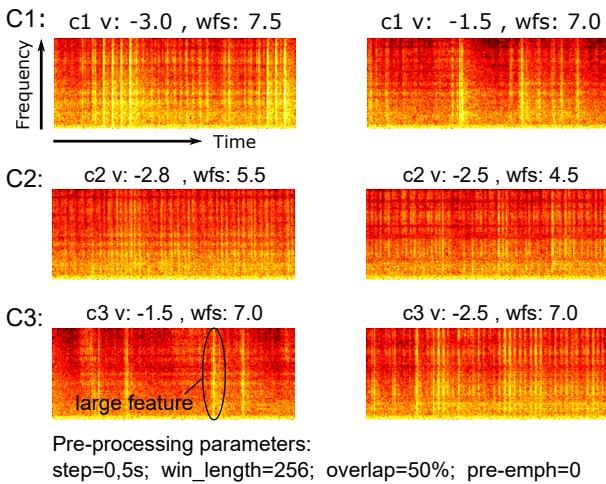


Figure 16: Six sample spectrograms generated at the same pre-processing parameters. The images, even within the same class, can be wildly different with irregularly spaced arc impulses (bright vertical lines), making it difficult to determine an optimal sample length (l_{step}). (The shown spectrograms span over 0,5s.)

As mentioned previously, the window length l_{win} determines the number of frequency bins the signal is sorted into. This comes however at a cost: A larger l_{win} increases the resolution in the frequency domain, but on the same time it stretches over a longer period in the time domain, averaging out any short-duration effects. Figure 17 illustrates this "uncertainty principle", (also known as the Gabor limit (Gabor, 1947)): the STFT either has accuracy in pinpointing the frequencies of the two harmonics, or temporal accuracy in locating the sharp clicks, but it cannot offer both. To partially compensate for the shortcomings of the STFT, the overlap can be increased, allow-

ing short-duration features to be present in more than one window, which makes them appear stronger on the spectrogram.

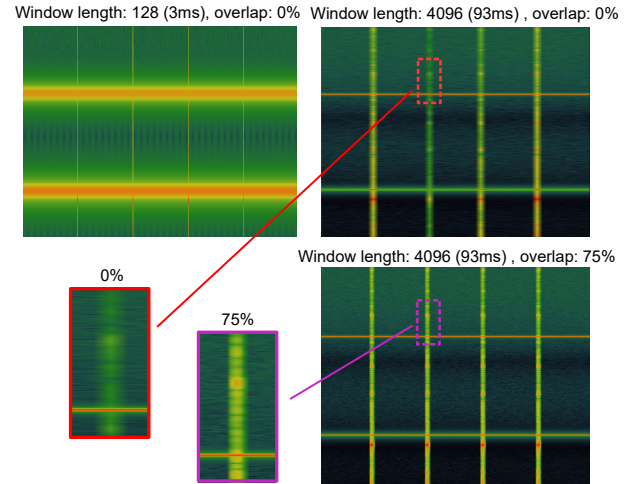


Figure 17: Spectrograms of a signal consisting of two sine waves at 5 and 15kHz, and four 5ms clicks. The various window lengths can either represent the waves or the clicks accurately, but not both. Increasing the overlap accentuates the short clicks as they appear in multiple time windows.

Some sources using CNNs for speech processing recommend boosting the input signal with a high pass filter (Fayek, 2016). A typical audio signal has an overall amplitude decrease with increasing frequency, (eg. human speech decreases with roughly -2dB/kHz , meaning the signal at 10 kHz will be $\approx 10x$ weaker), which can lead to inaccuracies in some implementations of the discrete Fourier transform (Tom Backstrom, Aalto University Wiki, 2019). Using a *pre-emphasis* filter balances the frequency spectrum, as well as may improve the Signal-to-noise ratio. In order to test that hypothesis, the sound is treated with a first-order filter before it is passed through the STFT (i.e. between step 2 and 3 in Figure 15) :

$$P(z) = 1 - \alpha \cdot z^{-1} \rightarrow y(t) = x(t) - \alpha x(t-1) \quad (5)$$

Where α is the pre-emphasis parameter, $P(z)$ is the frequency domain expression, $x(t)$ is the time series data, and $y(t)$ is the discrete implementation of the filter. The frequency response of the discrete filter for different pre-emphasis values is shown in Figure 18.

4.3. Pre-processing impact on CNN accuracy

With the knowledge on how the pre processing parameters affect the spectrogram it remains to be tested how they impact the CNNs performance. Due to the complex nature of the CNN algorithm, it is not possible to a priori assess which choice of pre-processing parameters will best accentuate the information that is relevant for correct classification. In order to analyze both their impact, as well as the possible interactions between them, the pre-processing parameters were tested in a *2k-factorial experiment*. In this method each of the parameters is varied in two settings and a response surface of the process

¹ The origin of the impulses is assumed to be from arc ignition, based on the study of arc behavior in (Horvat et al., 2011) and (M Čiudina, 2003)

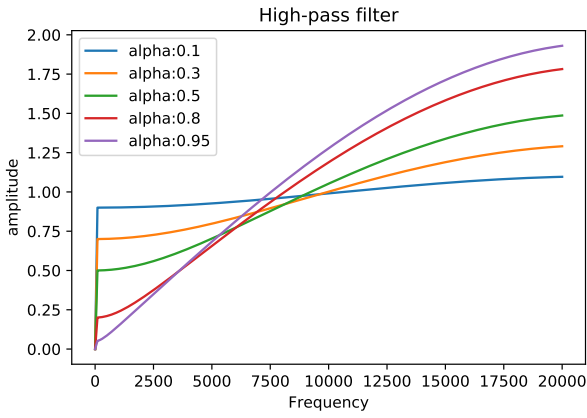


Figure 18: Frequency response of the discrete first-order filter from equation 5, for different values of α , at sampling frequency of 44kHz. (A more detailed analysis of the filter is available in the accompanying material)

is modelled, as illustrated in Figure 19. The method ensures to capture both the effect of the main parameters as well as their interactions, requires less experiments and is more robust towards finding the optimum than by optimizing the parameters one-at-a-time (Montgomery, 2009).

(A full walk-trough of the 2k-factorial experiment is available in the accompanying material, the explanation below serving as a resume.)

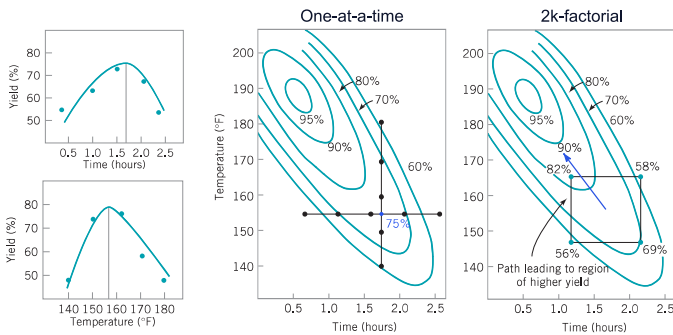


Figure 19: 2k-factorial experiment design on a process with two parameters. The 2k-factorial method uses only 4 experiments, and better characterizes the process than adjusting the factors one at a time, which would lead to a false optimum of 75%. (Montgomery, 2009) (Modified)

The four pre-processing parameters, l_{win} , l_{step} , $overlap$ and $pre-emph$ were encoded as the four variables $x_1 \dots x_4$ in the experiment and each assigned an empirical "high" and "low" level shown in Table 5. The four variables yield 16 unique combinations (called *treatments*). Each treatment was replicated twice with two different randomness seed values (0 and 100,) providing a total of 32 data points, shown in Figure 20.

In general it was found that there is weak statistical evidence for that the treatments have an impact on the recognition accuracy of the CNN, meaning that, within the span of the parameters chosen in table 5, it is difficult to distinguish any effect

| Parameter | low (-1) | high (+1) |
|-----------------|----------|-----------|
| $x_1: l_{step}$ | 0,25 | 0,5 |
| $x_2: l_{win}$ | 128 | 256 |
| $x_3: overlap$ | 25% | 50% |
| $x_4: preemph$ | 0,0 | 0,95 |

Table 5: Parameter values for the first 2k-factorial test.

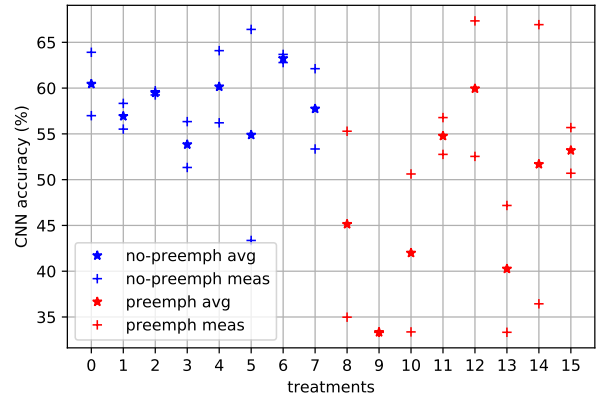


Figure 20: Results of the 16 treatments listed in Table 6

of a particular treatment from the overall variation in the set, as can partly be seen in the large overlaps between treatment results in Figure 20. More replicates of the treatments could partially help the issue, however due to the time constraints of the project it was decided to carry on with only 2 replicates of each treatment.

Secondly, it was early on established that pre-emphasis has a strong negative effect on the CNNs' performance as well as it would make it less stable, as plotted in Figure 21. (On average changing pre-emphasis from 0,0 to 0,95 would result in a 10,4 percentage-point (*pp*) drop in accuracy.) On those grounds it was decided to exclude pre-emphasis as a parameter, (i.e. discard the results of treatments 8 to 15) before continuing.

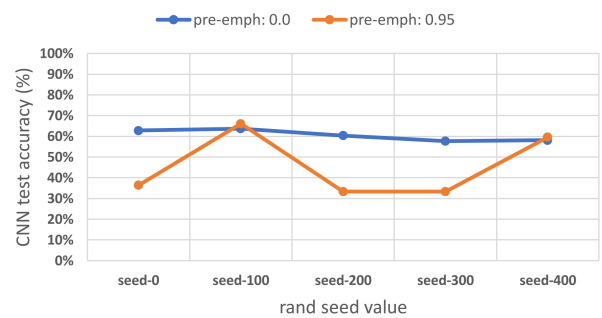


Figure 21: Comparison of 5 replicates of treatment no. 6 and 12, (see Table 6) highlighting the instability of the CNNs' performance when the pre-emphasis parameter is set high. (The seed value refers to the randomness seed used to generate the training set.)

After excluding the pre-emphasis the main effects and interactions of the remaining three parameters were identified, (for

| no. | x_1 | x_2 | x_3 | x_4 | (avg.) | y_0 | y_{100} |
|-----|-------|-------|-------|-------|--------|-------|-----------|
| 0 | -1 | -1 | -1 | -1 | 60.450 | 56.99 | 63.91 |
| 1 | +1 | -1 | -1 | -1 | 56.925 | 58.33 | 55.52 |
| 2 | -1 | +1 | -1 | -1 | 59.480 | 59.25 | 59.71 |
| 3 | +1 | +1 | -1 | -1 | 53.835 | 51.33 | 56.34 |
| 4 | -1 | -1 | +1 | -1 | 60.150 | 56.21 | 64.09 |
| 5 | +1 | -1 | +1 | -1 | 54.885 | 43.36 | 66.41 |
| 6 | -1 | +1 | +1 | -1 | 63.235 | 62.79 | 63.68 |
| 7 | +1 | +1 | +1 | -1 | 57.735 | 53.35 | 62.12 |
| 8 | -1 | -1 | -1 | +1 | 45.135 | 34.98 | 55.29 |
| 9 | +1 | -1 | -1 | +1 | 33.330 | 33.33 | 33.33 |
| 10 | -1 | +1 | -1 | +1 | 41.995 | 33.37 | 50.62 |
| 11 | +1 | +1 | -1 | +1 | 54.770 | 56.78 | 52.76 |
| 12 | -1 | -1 | +1 | +1 | 59.940 | 52.54 | 67.34 |
| 13 | +1 | -1 | +1 | +1 | 40.255 | 47.18 | 33.33 |
| 14 | -1 | +1 | +1 | +1 | 51.685 | 36.44 | 66.93 |
| 15 | +1 | +1 | +1 | +1 | 53.195 | 55.69 | 50.70 |

$y_0 = acc_u$ @ rand. seed 0 $y_{100} = acc_u$ @ rand. seed 100

Table 6: List of results for the 16 treatments in the first 2k-factorial run. The ± 1 values represent the parameter values defined in table 5.

| parameter | Δy (25/50%) | Δy (50/75%) |
|-------------------|---------------------|---------------------|
| x_1 | -4.98375 | -3.67875 |
| x_2 | 0.46875 | 2.57375 |
| x_3 | 1.32875 | 1.03125 |
| (x_1, x_2) | -0.58875 | -0.83375 |
| (x_1, x_3) | -0.39875 | 1.70375 |
| (x_2, x_3) | 2.49875 | -0.39375 |
| (x_1, x_2, x_3) | 0.47125 | -0.71625 |

Table 7: Main effects and interactions of the two 2k-factorial experiments. Δy is the average change (in *pp*) of the CNNs' acc_u if the listed parameter(s) are changed from their "low" to their "high" setting.

more details on the methodology refer to the accompanying material) shown on Table 7, and Figure 22.

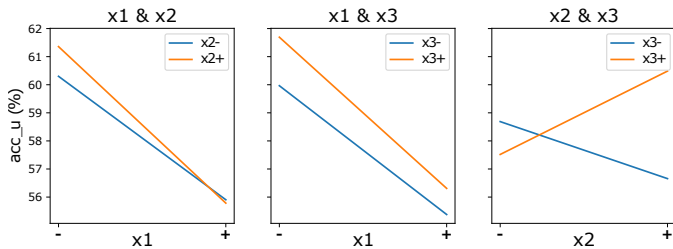


Figure 22: Plot of the 2-way interactions between the 3 parameters in the first iteration of the 2k-experiment. (See first column in table 7 for numeric values.)

From the effects and interactions it could be concluded that in order to obtain the best results: x_1 should be kept low. x_2 should be kept high, - not due to its importance on its own, but the impact when combined with x_3 and x_3 should be kept high.

Encouraged by the results, it was decided to conduct a series of further 4 treatments with the overlap increased to 75%. Combined with treatments no. 4 to 7 this would result in a new 2k-factor 3-parameter design. (I.e. the same experiment setup as listed in Table 5, but with the parameter values for x_3 being 50% and 75% respectively, and no variable x_4 .)

The second iteration yielded set of effects and interactions, listed in the second column of Table 7. In a similar fashion it was concluded that for best performance x_2 and x_3 should be kept high, while x_1 should be kept low, leading to treatment no. 6, ([1, 1, 1]) being chosen as the final set of processing parameters: l_{step} : 0,25s, l_{win} : 256, *overlap*: 75% and *pre-emph*: (none). The final parameters were then replicated 5 times, in order to test the stability of the results, yielding an average recognition accuracy of 61,13% and a standard deviation of 2,27 percentage-points, as shown in Figure 23, giving all in all an average improvement of 2,83*pp* over the average of iteration 1 (with x_4 excluded).

It has to be kept in mind that due to the high variation and low number of replicates in the 2k-experiments, all conclusions have to taken with a grain of salt, however some of the suggestions from the 2k-test can partly be explained: A high x_2 value (i.e. a high window length) results in more frequency bins of the STFT providing more resolution in the frequency domain. A high x_3 value, (i.e. a high overlap) accentuates the short-duration features, as illustrated in Figure 17. Interestingly those parameter values coincide with the ones chosen by (Ren et al., 2018).

The suggestion to reduce the step length is less intuitive: As illustrated in Figure 16, some of the files contain sharp features occurring irregularly, approximately once every 0,25s. The occurrence of those can itself not be linked to a particular class, as they occur in all 3 classes. The choice to keep the spectrogram length at 0,25s is partly motivated by the wish to enlarge the chance that on average, if they occur in the given file, at least one such feature will be present in each spectrogram. It could however be investigated if the strong arc impulses indeed contain information that can be linked to the penetration state, as claimed by e.g. (Ren et al., 2018).

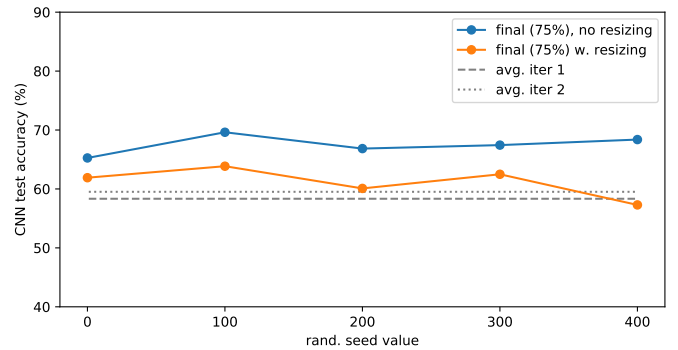


Figure 23: comparison of the Performance of the CNN with the final processing parameters, with and without image resizing, as well as the averages of the two iterations of the 2k factorial experiment.

4.4. Effect of image resizing

When the pre-processing parameters were varied during the 2k-factorial experiments, as a consequence of Equation 4, the resulting spectrograms had different image dimensions. Typically a CNN is designed to operate on a certain image size, as it affects the ratio of inputs (the number of pixels) to trainable parameters (filters and neurons in fully connected layers) in the CNN.

To avoid introducing this extra source of variation in the experiment, it was decided that all spectrograms in the 2k-factorial test would be resized to the same size. The size of the largest spectrogram size (129 x 345) was chosen and the images were resized using nearest-neighbor interpolation. Choosing the largest image size means the images are only enlarged, i.e. no information in the spectrogram would be lost and the choice of interpolation mode means that no additional data is created.

Of course, resizing the image still has an effect as some features on the image may no longer be able to be covered by the convolutional filter, as illustrated in Figure 24.

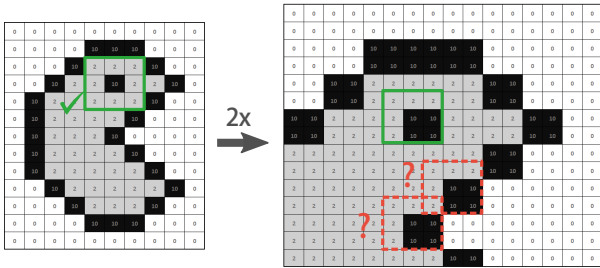


Figure 24: Effect of resizing: As the image is enlarged, the same 3x3 convolutional filter may be unable to unambiguously locate some of the features of the image. (e.g. the eye).

This possible effect of resizing can unfortunately not be accounted for during the 2k tests, however in order to characterize the effect of image resizing, the final design parameters were replicated 5 times without resizing the images, as shown in Figure 23. As shown on the figure, the un-resized spectrograms are achieving a better accuracy than their resized counterparts. For that reason, all experiments in the next sections are carried out on the chosen parameters, without image resizing. (i.e. an image size of 129 x 172.)

5. CNN Architecture

Having settled on a set of parameters to pre-process the input to the CNN, the network itself can now be put under scrutiny, in an attempt to design an optimal architecture for the task.

Numerous hyperparameters come into play when designing a CNN, both related to the "architecture" of the CNN itself such as the number, type, size and depth of the layers, as well as hyperparameters related to the training of the network: the choice of loss function, optimizer algorithm, learning rate or number of epochs to train. The correct choice of those hyperparameters, is the basis for achieving good algorithm performance, as

proven e.g. by (David Cox, 2011), achieving results ranging from pure chance to state-of-the-art performance, depending purely on hyperparameter choices.

Unfortunately, while there are methods for optimizing or pruning existing, functioning CNN networks (eg. (Garg et al., 2020)), very few systematic methodologies exist for manually designing and tuning a network from scratch and in many cases the choice of hyperparameters relies on domain knowledge of the researchers (Bergstra et al., 2011).

As the domain knowledge is very difficult to generalize across different application cases, most hyperparameter optimization algorithms that do exist, are based on random search (Bergstra & Bengio, 2012) or Sequential Model-Based Optimization (SMBO) being an implementation Bayesian model-based optimization (Ian Dewancker, 2015).

As evaluating the real objective function $f(x)$ (i.e. training and testing network at a given set of hyperparameters, x), is computationally expensive, the SMBO method works by building a probability-based surrogate model p , based on an initial dataset D of function evaluations at different hyperparameter values: ($D = \{(x_1, y_1), \dots, (x_n, y_n)\}$). Using the surrogate model as a fast alternative, a *selection function* S , can be optimized, which is used to decide what set hyperparameters would yield the improvement in the performance of f . The proposed set of new hyperparameters, x_i from the allowed search space X , can be seen as a "best educated guess based on current knowledge" and is tested on the real objective function. The new datapoint $y_i = f(x_i)$ is then added to D , which enriches the knowledge used for the next prediction. The procedure is then repeated for a set number of iterations, or time limit, as summarized in algorithm 1: (Bergstra et al., 2011), (Ian Dewancker, 2015)

Algorithm 1: Sequential Model-Based Optimization

```

input:  $f, X \supset x, S$ 
 $D \leftarrow \text{InitialSamples}(f, X)$   $\triangleright$  Create initial set of function evaluations
for  $i = |D|$  to  $i_{\max}$  do
   $p(y|x, D) \leftarrow \text{FitProbModel}(D)$   $\triangleright$  Fit surrogate model to  $D$ 
   $x_i \leftarrow \text{argmax}_x S(x, p(y|x, D))$   $\triangleright$  Optimize  $S$  based on  $p$ 
   $y_i \leftarrow f(x_i)$   $\triangleright$  Evaluate new  $x$ , (expensive)
   $D \leftarrow D \cup (x_i, y_i)$   $\triangleright$  Add new data-point to  $D$ 
end
 $x^*, y^* \leftarrow \text{max}(D)$   $\triangleright$  Retrieve the optimal parameters.

```

The Python *hyper-opt* (Bergstra et al., 2013) library offers an easy to use SMBO implementation, using Tree Parzen estimators (TPE) as the surrogate function, and a metric of Expected Improvement (EI), as the selection function. In trials on different computer vision problems, the algorithm tuned via the the TPE-method outperformed both ones optimized via. random search based methods as well as human experts.

Unfortunately, when applied on the current problem, due to the number of hyperparameters which can be tuned to improve the model, the method would require a high number of iterations, which is not possible due to hardware constraints².

². The mentioned TPE-based optimization required roughly 24 hours of

Similarly, if the hyperparameter space was to be searched using the 2k-factorial method as in Section 4.3, this would result in up to $2^6 = 64$ treatments in the first iteration alone, each requiring more than one replicate for reliable conclusions.

Facing those limitations it was decided to tune the hyperparameters manually, by first, in order to limit the search space, choosing a "base architecture" from a selection of different architecture types. Then, on the base architecture, 6 selected hyperparameters which could be of influence were investigated: The use of dropout layers, learning rate, choice of activation function, use of batch normalization and varying the kernel size.

The experiments implicitly assume no interactions between the hyperparameters, so that the parameters can be tuned sequentially. This assumption does not necessarily need to hold true, however is necessary in order to limit the search space.

5.1. Selecting a base architecture:

When designing a general CNN architecture it common practice to start with larger but shallower (i.e. fewer filters) convolutional layers, then as the network progresses the layers become smaller but with more filters. This is motivated by the fact that while the number of low level features is relatively low (e.g., vertical, horizontal lines, etc.), there are many different ways to combine them into higher level features. (Géron, 2019).

Following those rules of thumb, four candidates for a base architecture were selected:

5.1.1. Initial architecture: 01-arch

The tests done in sections 3 and 4 were done on an initial CNN architecture, used to investigate the effects of different sampling strategies and pre-processing parameters. This initial architecture, dubbed *01-arch*, is based on the network used by Ren et al. (2018) for classifying the penetration state of aluminium GTAW welding. (See Table 1) The architecture, listed below, bears resemblance to the LeNet-5 architecture (Lecun et al., 1998), by using two sets of Convolution-Pooling pairs followed by a fully connected layer. It characterizes itself by using an uneven stride length of (2,1) on the first convolutional layer, reducing the image height by half³ and by using Leaky-ReLU units as an activation function. As an addition to the original design, a dropout layer has been added after the last MaxPooling layer, with a dropout of 0.5 to reduce overfitting.

```
# 01-architecture:
alpha=0.01
model = Sequential([
    Conv2D(8, kernel_size=(3, 3), strides=(2, 1),
        ↪ padding='same',
        input_shape=input_shape),
```

computation on 6 parallel GPUs to reach human level performance. (Bergstra et al., 2013)

³. The input image used by Ren et al. was 129x32 pixels, therefore the uneven strides try to bring the closer to a 1:1 aspect ratio. Surprisingly, the stride length also works better on the current, 129x172 image than an equal stride of (2,2).

```
    LeakyReLU(alpha=alpha),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(16, kernel_size=(3, 3), strides=(2, 1),
        ↪ padding='same')),
    LeakyReLU(alpha=alpha),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.5),
    Flatten(),
    Dense(96),
    LeakyReLU(alpha=alpha),
    Dense(3, activation='softmax')
])
```

5.1.2. 02-arch

The second architecture is a carbon copy of the first, with the exception that it does not have its special characteristics: Normal ReLU activation functions are used⁴, and all convolutional layers have a normal stride length of (1,1). In order to test whether the improvement may lie in the fully connected layers, an extra dense layer was added and the total number of fully connected neurons was increased from 99 to 207:

```
# 02-architecture:
model = Sequential([
    Conv2D(8, kernel_size=(3, 3), activation='relu',
        padding='same', input_shape=input_shape),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(16, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(120, activation='relu'),
    Dense(84, activation='relu'),
    Dense(3, activation='softmax')
])
```

5.1.3. 03-arch

The third architecture is a variation of *02-arch*, except that more emphasis is put on the convolutional aspect of the network: instead of an extra dense layer, it has a third Convolution-pooling pair making it deeper. The number of filters is also dramatically increased, starting at 32 and, as suggested by Géron (2019), doubled after each Max pooling layer:

```
#03-architecture
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu',
        input_shape=input_shape),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
```

⁴. Motivated partly by their use in *01-arch*, as well as their superior performance to tanh activation functions in speech recognition tasks (Maas, 2013).

```

Flatten(),
Dense(64, activation='relu'),
Dense(3, activation='softmax')
)

```

5.1.4. 04-arch

The fourth architecture is inspired by the VGG-16 and VGG-19 networks (Simonyan & Zisserman, 2014) employing a Convolution-Convolution-Pooling structure, which is repeated twice. It also has a relatively large fully connected layer with 512 neurons.

```

#04-architecture
model = Sequential([
    Conv2D(8, kernel_size=(3, 3), activation='relu',
padding='same', input_shape=input_shape),
    Conv2D(8, kernel_size=(3, 3), activation='relu',
padding='same'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(16, kernel_size=(3, 3), activation='relu'),
    Conv2D(16, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(3, activation='softmax')
])

```

Tables summarizing the four architectures, their sizes and number of parameters can be found in Appendix C. Each of the four architectures was run 5 times on the same seed values, with the the average test accuracies listed in table 8.

Despite achieving the best results, *01-arch* is already a result of the work of (Ren et al., 2018), and is therefore not included in the further investigation. *04-arch* with the highest average testing accuracy of 60,46% is chosen as the best base architecture for further improvement.

| Architecture: | # params | avg acc _u (%) | std dev. (pp) |
|---------------------|------------|-----------------------------|------------------|
| 01-arch | 596.067 | 67,51 | 1,46 |
| 02-arch | 2.653.707 | 59,89 | 4,92 |
| 03-arch | 4.221.827 | 44,67 | 13,88 |
| 04-arch | 11.278.395 | 60,46 | 3,43 |
| 04-dropout | 11.278.395 | 64,66 | 3,16 |
| 04-drp-leaky0.1 | 11.278.395 | 63,90 | 3,64 |
| 04-drp-leaky0.3 | 11.278.395 | 63,27 | 1,14 |
| 04-drp-leaky0.3-SGD | 11.278.395 | 67,46 | 1,96 |

Table 8: Number of trainable parameters, the average test accuracy and standard deviation (over 5 repetitions, seed values as in Figure 27) of the four baseline candidates and the further versions of *04-arch*.

5.2. Architecture-related hyperparameters

With a baseline architecture selected its different "internal" hyperparameters can be varied. This section emphasizes on hyperparameters acting as modifiers to the base architecture.

As some of the parameter choices may not work at all, the tests in this section were conducted with *early stopping* enabled and a patience value of 2, i.e. if the validation loss increases for more than 2 consecutive epochs, it is interpreted as an onset of model overfitting, and the training is stopped and lowest validation loss so far is saved, as illustrated in Figure 25.

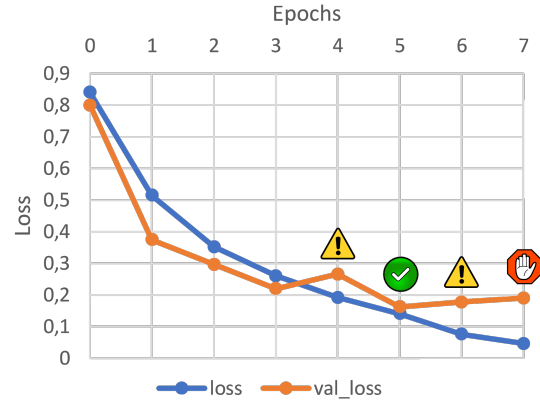


Figure 25: Early stopping with patience = 2: If the validation loss does not decrease within 2 consecutive epochs, the training is terminated, and the model at the lowest validation loss is ket. This reduces the risk of overfitting, which is assumed to begin once the disparity between training and validation loss starts to diverge. (i.e. epoch 5).

5.3. Dropout

The first modification to the base architecture was the addition of a dropout layer after the last convolutional max pooling, and before the fully connected layers. (I.e. between layers P6 and FL7, see table C.12 in Appendix C). Dropout regularization works by at any given training step giving each neuron and in the regularized layer a set probability of being turned off, i.e. its value and connections being ignored. (Srivastava et al., 2014)

This simple method prevents the network from having complex co-adaptations, i.e. specialized neurons that activate based on very specific patterns in the input, but do not generalize well when exposed to unseen data. (Hinton et al., 2012)

Dropout rates of (0.2, 0.4, 0.6, 0.8) were tested, and a dropout rate of 0.6 proved to yield the best test performance, increasing the average acc_u from 60,46% to 64.66% as listed in Table 8. The change was therefore kept and incorporated into the network.

A second dropout layer was also inserted between the two dense layers (I.e. between layers D8 and D9 in table C.12) and tested with the same values, however no improvement in performance was observed.

5.3.1. Leaky - ReLU

As a second modification, based on its use in *01-arch*, it was tried to use the Leaky Rectified Linear Units (Leaky-ReLU), activation function, which in differs itself from the regular ReLU function, by having a nonzero slope when the unit is not active (Chollet et al., 2015).

$$f(x) = \begin{cases} \alpha \cdot x & , x < 0 \\ x & , x \geq 0 \end{cases} \quad (6)$$

Where α is the slope parameter. The small slope, set by α , reduces a problem known as the *dying ReLUs* where the weighted sum of all inputs to a neuron is negative for in all instances of the training set and as illustrated in Figure 26, the neuron does not pass on any signal. Leaky-ReLUs also partly reduce the *vanishing gradients* problem. (Géron, 2019).

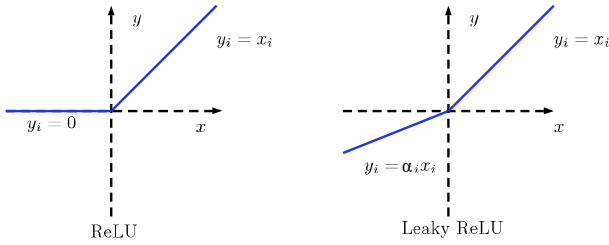


Figure 26: ReLU and Leaky-ReLU activation functions. (Xu et al., 2015)

Similar to *01-arch*, all activations were changed from ReLU to Leaky-ReLU. Based on suggestions from (Xu et al., 2015) and (Han et al., 2016) an alpha of 0,1 and 0,3 were tested, yielding an average acc_u of 63,9% and 63,27% respectively. While those numbers are one percentage-point lower than the previous *04-dropout* architecture listed in 8, the runs with an α of 0,3 had a much lower standard deviation of just 1,14pp, as shown in Figure 27. As a stable performance is preferable in the further search for hyperparameters over a small drop in accuracy, the change is kept.

5.3.2. Batch normalization

Generally, in deep neural networks training is complicated by *internal co-variate shift* (the fact that the distribution of each layers input changes as the parameters of the previous layer change during training, so that small changes in the parameters amplify as the network becomes deeper). Batch normalization (BN) addresses the problem by, during training, zero-centering and normalizing the input before/after passing it on to the activation function. The input is normalized based on the mean and standard deviation of the current mini-batch, hence the name. (Géron, 2019) (page 333.) (Ioffe & Szegedy, 2015)

Although typically used in deep neural network architectures, BN can also be used in CNNs. Similar to its use in e.g. *ResNet* (He et al., 2015), three BN layers were inserted in the architecture, before each MaxPooling and Dropout layer, (i.e. before layers P3, P6 and D8, in table C.12) and tested on batch sizes of 32 (default), 128 and 256. No improvements were noted in either the test, or training performance of the networks, quite contrary, the networks were found to have poor learning performance, resulting in early stopping after just 2-3 epochs.

Similar results were noted with BNs inserted before every non-linearity in the model, (i.e. before every layer with a Leaky-ReLU activation) as suggested by the creators in their original publication. (Xu et al., 2015)

5.3.3. Kernel size

A last hyperparameter that was varied was the size of the convolutional kernel. As discussed previously in Section 4.4 the kernel size may influence what features in the image the CNN is able to capture. In order investigate that assumption, the first convolutional layer, C1 was modified to use a 5x5 filter with a stride length of (2,2). When implemented the change yielded poor results, with an average acc_u of 45,7%, and in 3 out of the 5 cases the network did not start learning at all.

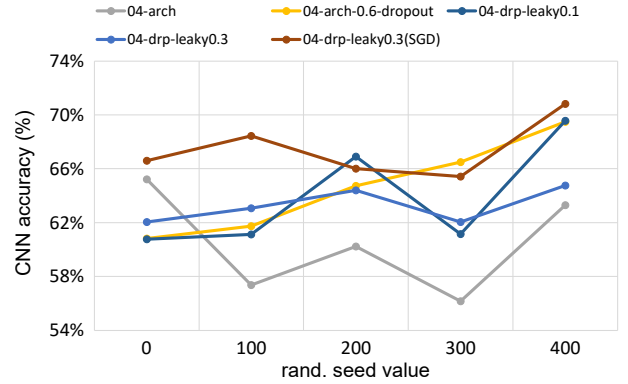


Figure 27: Test performance of the 4 variants of *04-arch* listed in Table 8. The seed values refer to the randomness seed values used to generate the training dataset.

5.4. Training hyperparameters

Besides the hyperparameters related to the design of the network itself, it is important to consider the training method, as it determines whether the network is able to reach its full potential.

5.5. Learning rate

The learning rate can be a hugely important hyperparameter as it can have influence on both the convergence speed as whether the optimizer is capable of converging to the minimum at all, or may be stuck at local plateaus. Typically the inputs to a model are normalized so a default learning rate of 0,01 would work for most problems (Montavon et al., 2012) (page 437.), nevertheless the best learning rate may depends on the particular problem, and should therefore always be reassessed. As learning rates typically lie in the interval 10^{-6} to 1 (Montavon et al., 2012) (page 437.) Common advice (Surmenok, 2017), (Ramesh, 2018) suggests to try learning rates of sequentially increasing orders of magnitude: i.e. 0,001, 0,01, 0,1, 1. The suggested learning rates of 0,001 (default), 0,01 and 0,1 were tried, however no significant improvements, neither in accuracy nor training time were observed and the default rate of 0,001 for the *Keras Adam* optimizer was kept.

5.5.1. F1-loss

Until now, all networks were trained using the *Categorical Cross-entropy loss* which is used as the the default in *Keras* multi-class problems. However, due to the imbalance in the representation of the classes in the dataset, as discussed in Section

2.2 other loss functions may be more appropriate. (Note that the 3 classes are still equally represented in the training data, but due lack of raw c_1 data, the c_1 samples are more similar, as discussed in Section 3.)

The F1-score is an expression of the balance between *Precision* and *Recall* of a set of predictions, defined as:

$$\begin{aligned}
 Precision &= \frac{TP}{TP + FP} \\
 Recall &= \frac{TP}{TP + FN} \\
 F1 &= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}
 \end{aligned}
 \tag{7}$$

Where TP , FP and FN are true positives, false positives and false negatives, respectively (see Figure 28.)

The precision metric asks the question: "what fraction of the predicted positives were correct?", favoring only classifying "sure bets" as positive, while the recall asks: "what fraction of actual positives was caught by the algorithm?" favoring a "label as positive, just in case" approach. The F1 score balances the two, favoring classifiers that have similar precision and recall.

The F1-score cannot be used directly as a loss function, as it is not differentiable for back-propagation. This can be changed by slightly modifying it, so instead of the 0/1 predictions the probabilities from the softmax layer are accepted:

Algorithm 2: F1-loss

input: $y, \hat{y}, thresh$

$\hat{y} = greater(\hat{y}) \quad \triangleright$ set $y_i < thresh$ to 0

$tp = \sum_{i=1}^c (y_i \cdot \hat{y}_i)$

$fp = \sum_{i=1}^c ((1 - y_i) \cdot \hat{y}_i)$

$fn = \sum_{i=1}^c (y_i \cdot (1 - \hat{y}_i))$

$p = \frac{tp}{tp + fp + \epsilon}$

$r = \frac{tp}{tp + fn + \epsilon}$

$F1 = \frac{2 \cdot p \cdot r}{p + r + \epsilon}$

return $1 - F1 \quad \triangleright$ We minimize $1 - F1$ i.e. maximize $F1$

Where c is the number of classes, y is the one-hot encoded vector with the true labels, $y = [0, 0, 1]$ and \hat{y} is the softmax predictions ranging from 0 to 1. ϵ is a machine epsilon added to avoid division by zero. The input predictions can also be optionally filtered through a threshold filter (typically set to 0,5). (Maiza, 2019)

| | | Predicted | |
|--------|----------|----------------|----------------|
| | | Negative | Positive |
| Actual | Negative | True Negative | False Positive |
| | Positive | False Negative | True Positive |

Figure 28: Definition of terms in equation 7.

The F1-loss was tried with the current best network, (*04-drp-leaky0.3*, see table 8) at various threshold values. No sig-

nificant improvement in neither test accuracy or training performance was noted.

5.5.2. Choice of optimizer

After the endeavours of the previous sections, one last hyperparameter remaining is the choice of the optimizer itself. Until now the Keras *Adam* optimizer was used, as it is often considered a good all-purpose optimizer for a wide range of problems. However, a suggestion in (Géron, 2019) p.351 points to a paper by (Wilson et al., 2017) showing that adaptive gradient optimizers (eg. RMSProp and Adam) on over-parametrized datasets may find solutions which generalize poorly in testing. Following the suggestion, a *Stochastic Gradient Descent* (SGD) optimizer was used instead, with a Nesterov Accelerated Gradient (NAG), explained in Figure 29.

Different values of the momentum parameter were experimented with, settling on a $m = 0,9$ and a fixed learning rate of 0,01.

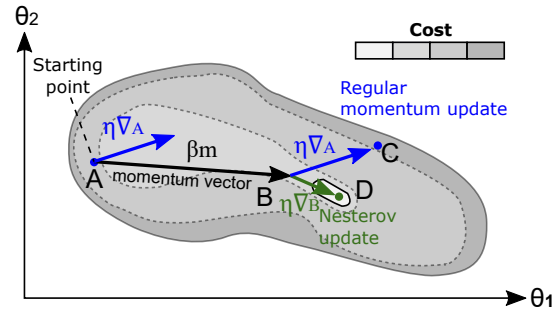


Figure 29: Nesterov Accelerated Gradient: Normally the optimizer would take a step in the direction of the gradient at point A plus the momentum from previous steps ($\vec{s} = \eta \nabla_A + \beta m$), NAG instead calculates the gradient at the end of the momentum vector, Point D, ($\vec{s} = \eta \nabla_B + \beta m$), which brings it closer to the optimum, (and less likely to overshoot.) (Géron, 2019)(modified)

As the SGD by nature has a more erratic path of descent, its validation loss also in the first few epochs shows more of a zigzag pattern, when compared to Adam as shown in Figure 30. As this is expected, the patience factor for early stopping was increased to 4. The SGD also generally converges slower, and the maximum number of epochs was likewise extended from 20 to 50.

As seen on the figure, training with the SGD optimizer took longer, but it did improve the test accuracy by over 4 percentage points, (from 64,7% to 67,5%, listed as *04-drp-leaky0.3-SGD* on table 8), without compromising the stability. If the average drop between validation accuracy and testing accuracy, (i.e. a hint on how well the model generalizes to the unseen test set) is calculated, the Adam optimizer has an average accuracy drop of 16,25pp over the 5 trials, whereas SGD has a slightly lower drop of 15,12pp. The difference is too small to firmly conclude that the model trained on the SGD is able to generalize better, nevertheless the small difference, coupled with the better performance of the SGD could motivate a deeper look into the differences between adaptive optimizers and SGD in future work.

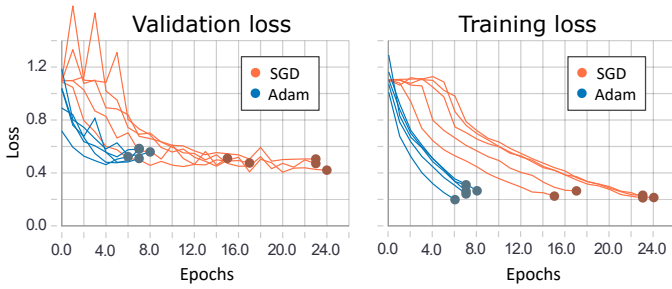


Figure 30: Plot of 5 runs with the same seed values for the SGD and Adam optimizer respectively. While SGD converges slower and with more erratic validation loss, SGD achieves better results in testing.

The *04-arch* using SGD with the added dropout and Leaky-ReLU is considered the final version of the CNN, identified as *04-drp-leaky0.3-SGD* its Keras definition shown below:

```
#04-drp-leaky0.3-SGD (Final architecture)
alpha = 0.3
model = Sequential([
    Conv2D(8, kernel_size=(3, 3), padding='same',
        input_shape=input_shape),
    LeakyReLU(alpha=alpha),
    Conv2D(8, kernel_size=(3, 3), padding='same'),
    LeakyReLU(alpha=alpha),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Conv2D(16, kernel_size=(3, 3)),
    LeakyReLU(alpha=alpha),
    Conv2D(16, kernel_size=(3, 3)),
    LeakyReLU(alpha=alpha),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.6),
    Flatten(),
    Dense(512),
    LeakyReLU(alpha=alpha),
    Dense(3, activation='softmax')
])
```

6. Testing the final architecture

In section 3 it was concluded that a $n_{times} = 2$ (i.e. a 2x oversampling of class c_1) was a good trade-off between a good performance and shorter training time. With the CNN model set, *04-drp-leaky0.3-SGD* was tested on a 2x oversampling on the *whole dataset* (i.e. a $n_{times} = 6$), in order to produce more training samples. The over 5 training runs, the model yielded, an average acc_u of 69,94% with a standard deviation of 2,14pp, confirming the earlier conclusion of Section 3, that a larger oversampling is still beneficiary for testing performance.

The greater overall number of samples, also means that the optimizer is able to see more examples in each epoch, meaning that its validation loss behaves more smoothly than previously illustrated in Figure 30.

Lastly, as discussed in Section 2.3, a second dataset *v35* was created consisting only of welding experiments at the same welding voltage. The hypothesis was that this would remove an extra noise factor from the dataset, perhaps leading to better results. As the *v35* dataset is much smaller the model was again trained with 2x oversampling of the whole dataset, giving an average acc_u of 70,17%. Surprisingly neither the average testing performance, nor the class accuracy of the individual classes look much different from models trained on the whole dataset, suggesting that the "training value" of the two data sets is similar.

6.0.1. Confusion characteristics

A confusion matrix of the final CNN model, is seen in Figure 31.

| | | Predicted | | | |
|-----------------|----|-----------|-------|-------|--|
| | | c1 | c2 | c3 | |
| Actual | c1 | 50,3% | 6,7% | 3,3% | |
| | c2 | 37,6% | 81,1% | 34,2% | |
| | c3 | 12,1% | 12,2% | 62,6% | |
| Prediction acc. | | 50,3% | 81,1% | 62,6% | |

| | | Predicted | | | Class acc. |
|--------|----|-----------|-------|-------|------------|
| | | c1 | c2 | c3 | |
| Actual | c1 | 65,9% | 18,0% | 16,1% | 65,9% |
| | c2 | 11,3% | 50,2% | 38,5% | 50,2% |
| | c3 | 4,4% | 9,2% | 86,3% | 86,3% |

Figure 31: Confusion matrix for the *04-drp-leaky0.3-SGD* (average of 5 models). The upper matrix shows prediction accuracies (columns add up to 100%), the lower, the class accuracies.)

Looking at the class accuracies (rows of lower matrix) it is clear that the 3 classes are "lying on a line", as illustrated conceptually in Figure 32: If the true class is c_3 the sample is more likely to be misclassified as c_2 than c_1 , (and vice versa for c_1) which hints that c_2 lies between c_1 and c_3 . This makes logical sense, as during the welding process, the category cannot change from c_1 (lack of penetration) to c_3 (excessive penetration), without going trough c_2 (normal penetration).

Secondly, if the true class is c_2 the sample is more likely to be misclassified as c_3 rather than c_1 , hinting that class c_2 resembles c_3 more than c_1 , hence the unequal distance illustrated in Figure 32. This corresponds well with the predictions of the PCA analysis in Section 2.3, particularly in Figure 6, showing a bigger overlap between the points of c_2 and c_3 .

Comparing the prediction accuracy with the class accuracy, it can be hypothesized how the network weights the different classes, as drawn in Figure 32: If the 3 classes are seen as 3 distinct agents, a column of the class accuracy confusion matrix, can be seen as a measure of the probability of the class "laying a bet" on a sample, given the real class it belongs to. E.g. as shown on the upper part of the figure, $p(\hat{c}_1|c_1) = 0,659$, $p(\hat{c}_1|c_2) = 0,113$ and $p(\hat{c}_1|c_3) = 0,04$.

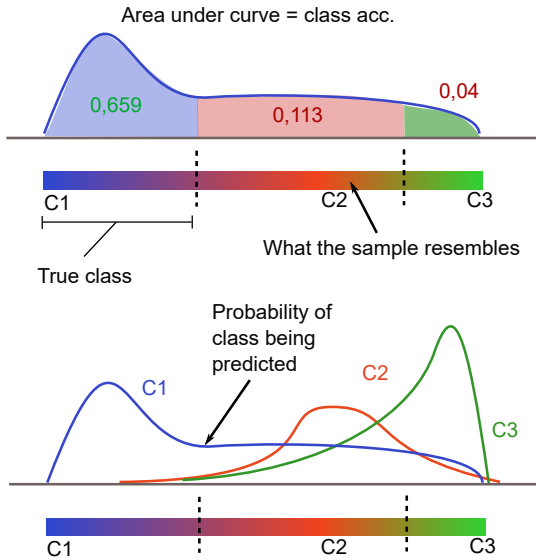


Figure 32: The confusion matrix conceptually drawn as a the hypothesized "selection probability curve" of the 3 classes.

The probability of that bet being right, is then the prediction accuracy, which can be used to approximate the shape of the curve: E.g. for c_2 : $p(\hat{c}_2|c_2) = 0,50$ meaning there is only a small chance of c_2 being predicted, but when the network decides to bet on it, the prediction accuracy is high (81,1 %), meaning that the peak of the curve is probably well centered on the class.

7. Conclusions and future work

In this article the use of Convolutional Neural Networks (CNNs) for the task of classifying the weld penetration state via the process sound was investigated.

An initial analysis of the obtained welding sound dataset in Section 2.3 revealed that the characteristics of the sound may vary depending on not only the penetration state, but also the welding parameters used and that the sound of the 3 penetration states is highly similar, requiring more than a PCA analysis of sound features to reliably separate out the 3 states.

In Section 4 the optimal pre-processing parameters for converting the sound time series data to spectrogram images were investigated. It was found that in general, the use of pre-emphasis has a strong negative influence on the usability of the spectrograms as input to the CNN. Secondly, the remaining pre-processing parameters have a statistically weak influence on the CNN performance, with a short step length and high overlap having a mildly positive effect.

With the pre-processing parameters set at the recommended values, a custom CNN architecture was built and manually tuned with various iterations of the design, achieving a gradually increasing testing accuracy, shown on figure 33, culminating in a test accuracy of 69,94% with a standard deviation of 2,14 percentage points. The improvements from the baseline accuracy of 60,46% were achieved through the use of Dropout, Leaky-ReLU activation functions and a Stochastic Gradient Descent

(SGD) optimizer. The final trained model is relatively lightweight being able to classify a 129x172 spectrogram in 0,0025s, i.e. 1/10th of the time that the image represents while running on a regular *Nvidia GTX 960M* laptop GPU⁵. This makes it feasible to envision it in the end application being used on e.g. an industrial computer.

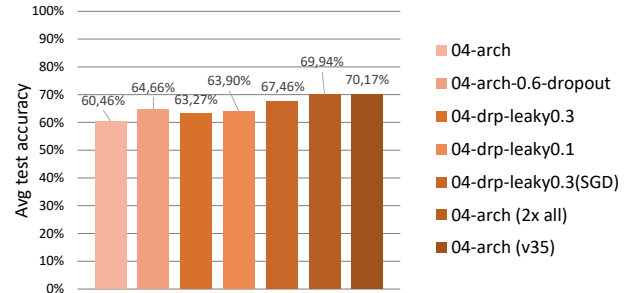


Figure 33: Testing accuracy of the various iterations of *04-arch*. *04-arch (2x all)* and *04-arch (v35)* refer to the tests described in Section 6

7.1. Discussion and recommendations for future work

While the final accuracy of the model is not sufficient for applying it to online weld monitoring we are still firmly convinced that, once properly implemented, CNNs can be used as a powerful tool for the application.

7.1.1. Recommendations for further experiments

Firstly, some of the poor performance of the CNN can be traced back to the dataset itself. As discussed in Section 2.3 the varying welding parameters may be an additional source of noise in the used dataset. For future work it could therefore be recommended that all welds are performed with a constant set of process parameters and the different penetration states are provoked by varying the workpiece (e.g. the root gap) or torch position, as it more likely would occur in an industrial process. (It could also be considered to use aluminium as material, as it is much more difficult to weld, making it easier to provoke different process faults.) Furthermore, based on the literature study in Section 1 there seems to be a significant difference in the abundance of information contained in the sound in AC and DC welding. Unless the DC welding is of particular interest, implementing CNN's for weld monitoring should be much easier for AC. It has also been speculated by the authors, whether placing a second microphone on the underside of the weld bead may be useful for detecting lack of penetration.

7.1.2. Considerations for further hyperparameter tuning

Besides the faults of the dataset, the CNN could also be improved by further tuning of hyperparameters. As discussed in section 5, provided the computational resources, the model lends itself to automated hyperparameter optimization, which

⁵ - Note that the computational time required to create the spectrogram is not considered.

would be able to adjust all the discussed hyperparameters (including the 4 base architectures) simultaneously.

In Section 5.3.2 the use of Batch normalization was discussed, and its poor performance when applied to the network. In retrospective BN assumes that the samples within each batch are independent and identically distributed. In theory no such guarantee is given for the spectrograms and an analysis whether this is indeed the case may shed some light on why this otherwise popular regularization method failed so badly.

Throughout the tests in Section 5, the various versions of the CNN has always scored lower validation and testing accuracies on the c_2 class, eg. illustrated on figure 31. This leads to believe that the c_2 examples are more difficult to classify than the remaining classes. The *Focal loss* (FL) function developed by (Lin et al., 2017) addresses just that.

The normal Cross entropy (CE) loss function, assigns a low loss value to well-predicted examples and then a progressively higher loss to less well-predicted. By adding a modifier term, $(1 - p_t)^\gamma$ the Focal loss is able to reduce the loss for the "easy to guess" examples, as shown on Figure 34. This means that with eg. at a p_t of 0,9 and a $\gamma = 2$, the a sample would get a 100x lower loss of 0,001 compared to 0,1 loss with CE. (In the meantime an incorrectly classified sample of $p_t = 0.45$ would only have a loss reduction of about 3,3x)

This choice of loss function accelerates learning in scenarios where the dataset contains highly imbalanced classes, as the FL function down-weights the relative impact across of the well-learned examples ($\sim 0,6 < p_t < 1$).

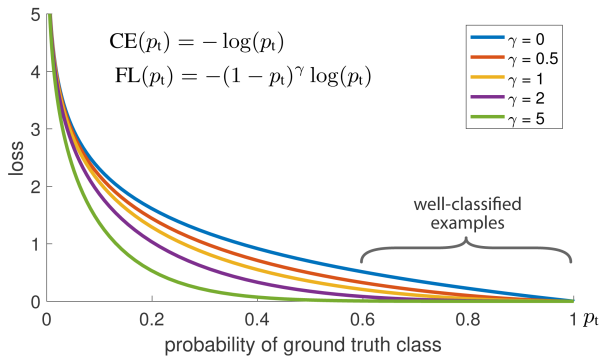


Figure 34: Focal loss as a function of γ . (Lin et al., 2017) At $\gamma = 0$ Focal loss becomes CE-loss (blue line). The figure is for a binary classification, i.e. a $p_t > 0,5$ means the class is predicted correctly.

The FL function can further be modified with an class weighting factor α_t so that the loss function becomes:

$$-\alpha_t \cdot (1 - p_t)^\gamma \cdot \log(p_t) \quad (8)$$

α_t can then be defined for arbitrarily for additionally weighing certain classes. (in case of unbalanced datasets, it could e.g. be set to the inverse class frequency.)

One last question which remains unanswered is how the ANN used by Bidstrup (2017), which operated on the exact

same dataset was capable of achieving 80% testing accuracy on a similar time step of 0,25 and 0% overlap, like test procedure used by the authors. A part of the answer may lie within the fact that the spectrogram images used by the CNN operate only on frequency-domain data, while Bidstrups input to the ANN included temporal and features, which simply do not exist in the spectrogram. A look at the literature review in Table 1 reveals that some temporal features (eg. signal kurtosis) recur in many of the publications, implying that they may great predictors of penetration state.

Furthermore the total number of features extracted by the various authors is fairly limited, ranging from 10-20. This means that some of those temporal features could in theory be extracted from the sound alongside the spectrogram and added to it as a few additional rows of pixels. While this defeats the main purpose of using spectrograms in the first place (avoiding feature engineering,) it could be interesting to see if this best-of-both-worlds approach could yield better results at a slightly higher computational cost.

Acknowledgements

The authors of this paper would like to thank Anders Bidstrup and prof. Simon Bøgh for their invaluable help during the process of creating this thesis.

References

- Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., 2011. Algorithms for hyper-parameter optimization. En: Proceedings of the 24th International Conference on Neural Information Processing Systems. NIPS'11. Curran Associates Inc., Red Hook, NY, USA, p. 2546–2554. URL: <http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>
- Bergstra, J., Bengio, Y., Feb. 2012. Random search for hyper-parameter optimization. The Journal of Machine Learning Research 13 (1), 281–305.
- Bergstra, J., Yamins, D., Cox, D. D., 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. En: Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28. ICML'13. JMLR.org, p. I-115–I-123.
- Bidstrup, A., 2017. Acoustic sensing for metal transfer mode and penetration state classification of gmaw using artificial neural networks. Master's thesis, Aalborg University.
- Cary, H. B., 2004. Modern welding technology., 6th Edición. Prentice hall, Upper Saddle River, N.J.
- Chollet, F., et al., 2015. Keras. <https://keras.io>.
- David Cox, N. P., 2011. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. En: Face and Gesture 2011. pp. 8–15.
- Dong, X., Wen, G., Ren, W., Luan, R., Yang, Z., Zhang, Z., 2017. Frequency selection for on-line identification of welding penetration through audible sound*. En: 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER). pp. 326–331. DOI: 10.1109/CYBER.2017.8446299
- Fayek, H., 2016. Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between. URL: haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html
- Gabor, D., May 1947. Acoustical quanta and the theory of hearing. Nature 159 (4044), 591–594. DOI: 10.1038/159591a0

- Garg, I., Panda, P., Roy, K., 2020. A low effort approach to structured cnn design using pca. *IEEE Access* 8, 1347–1360.
- Guu, A. C., Rokhlin, S. I., 1990. Weld penetration control with radiographic feedback. *Review of Progress in Quantitative Nondestructive Evaluation* 9, 1973–1980.
- Géron, A., 2019. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edición. O'Reilly Media.
- Han, Y., Kim, J., Lee, K., May 2016. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *arXiv e-prints*, arXiv:1605.09507.
- He, K., Zhang, X., Ren, S., Sun, J., Dec. 2015. Deep Residual Learning for Image Recognition. *arXiv e-prints*, arXiv:1512.03385.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. R., Jul. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv e-prints*, arXiv:1207.0580.
- Horvat, J., Prezelj, J., Polajnar, I., Čudina, M., 2011. Monitoring gas metal arc welding process by using audible sound signal. *Strojniški vestnik - Journal of Mechanical Engineering* 57 (3), 267–278. DOI: 10.5545/sv-jme.2010.181
- Ian Dewancker, Michael McCourt, S. C., 2015. Bayesian optimization primer. Tech. rep., SigOpt. URL: https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf
- Ioffe, S., Szegedy, C., Feb. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv e-prints*, arXiv:1502.03167.
- Jolly, W., 1969. The use of acoustic emission as a weld quality monitor. Tech. Rep. BNWL-SA-2727, Battelle Memorial Institute Pacific Northwest Laboratory.
- Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 12 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 2278 – 2324. DOI: 10.1109/5.726791
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P., Aug. 2017. Focal Loss for Dense Object Detection. *arXiv e-prints*, arXiv:1708.02002.
- Liu, X., Wu, C., Jia, C., Zhang, G., 2017. Visual sensing of the weld pool geometry from the topside view in keyhole plasma arc welding. *Journal of Manufacturing Processes* 26, 74 – 83. DOI: <https://doi.org/10.1016/j.jmapro.2017.01.011>
- Lv, N., Xu, Y. L., Fang, G., Yu, X. W., Chen, S. B., 2016. Research on welding penetration state recognition based on bp-adaboost model for pulse gtaw welding dynamic process. En: 2016 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO). pp. 100–105.
- Lv, N., Zhong, J., Chen, H., Lin, T., Chen, S., 2014. Real-time control of welding penetration during robotic gtaw dynamical process by audio sensing of arc length. *The International Journal of Advanced Manufacturing Technology* 74 (1), 235–249. DOI: 10.1007/s00170-014-5875-7
- M. Čudina, J. Prezelj, I. P., 2008. Use of audible sound for on-line monitoring of gas metal arc welding process. *Metalurgija* 47, 81–85.
- M Čludina, J. P., 2003. Evaluation of the sound signal based on the welding current in the gas-metal arc welding process. *Proceedings of the Institution of Mechanical Engineers ,Part C: J. Mechanical Engineering Science* 217, 483–494.
- Maas, A. L., 2013. Rectifier nonlinearities improve neural network acoustic models.
- Maiza, 2019. The unknown benefits of using a soft-f1 loss in classification systems. URL: <https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d>
- Michel Misiti, Y. M., 1996. *Wavelet Toolbox User's Guide*. MathWorks, 24 Prime Park Way, Natick, MA 01760-1500, 1st Edición.
- Montavon, G., Orr, G. B., Müller, K.-R., 2012. *Neural Networks: Tricks of the Trade*, 2nd Edición. Springer, Berlin, Heidelberg.
- Montgomery, D. C., 2009. *Introduction to Statistical Quality Control*, 6th Edición. John Wiley & Sons.
- Pal, K., Bhattacharya, S., Pal, S. K., 2009. Prediction of metal deposition from arc sound and weld temperature signatures in pulsed mig welding. *The International Journal of Advanced Manufacturing Technology* 45 (11), 1113. DOI: 10.1007/s00170-009-2052-5
- Pal, K. P. S. K., 2011. Monitoring of weld penetration using arc acoustics. *Materials and Manufacturing Processes* 26:5, 684–693.
- Ramesh, S., 2018. A guide to an efficient way to build neural network architectures- part i: Hyper-parameter selection and tuning for dense networks using hyperas on fashion-mnist. URL: <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>
- Ren, W., Wen, G., Liu, S., Yang, Z., Xu, B., Zhang, Z., 2018. Seam penetration recognition for gtaw using convolutional neural network based on time-frequency image of arc sound. En: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). Vol. 1. pp. 853–860. DOI: 10.1109/ETFA.2018.8502478
- Rostek, W., 1990. Investigation on the connection between the welding process and airborne noise emission in gas-shielded metal-arc welding. *Schweissen Schneiden* 6.
- Saad, E., Wang, H., Kovacevic, R., 2006. Classification of molten pool modes in variable polarity plasma arc welding based on acoustic signature. *Journal of Materials Processing Technology* 174 (1), 127 – 136. DOI: <https://doi.org/10.1016/j.jmatprotec.2005.03.020>
- Saini, D., Floyd, S., April 1998. An investigation of gas metal arc welding sound signature for on-line quality control. *Welding Research Supplement*, 172–179.
- Simonyan, K., Zisserman, A., Sep. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, arXiv:1409.1556.
- Sipei Zhao, Xiaojun Qiu, I. B. M. R., Lele, A., 2018. Gmaw metal transfer mode identification from welding sound. En: *Hear to Listen - Proceedings of ACOUSTICS 2018*.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15 (56), 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>
- Surmenok, P., 2017. Estimating an optimal learning rate for a deep neural network. URL: <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>
- Tarn, J., Huissoon, J., 01 2005. Developing psycho-acoustic experiments in gas metal arc welding. Vol. 2. pp. 1112 – 1117 Vol. 2. DOI: 10.1109/ICMA.2005.1626707
- Tom Backstrom, Aalto University Wiki, 2019. Pre-emphasis. URL: <https://wiki.aalto.fi/display/ITSP/Pre-emphasis>
- Wang, J. F., Yu, H. D., Qian, Y. Z., Yang, R. Z., Chen, S. B., 2011. Feature extraction in welding penetration monitoring with arc sound signals. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 225 (9), 1683–1691. DOI: 10.1177/0954405411405108
- Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., Recht, B., May 2017. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *arXiv e-prints*, arXiv:1705.08292.
- Wu, D., Huang, Y., Chen, H., He, Y., Chen, S., 2017. Vppaw penetration monitoring based on fusion of visual and acoustic signals using t-sne and dbn model. *Materials & Design* 123, 1 – 14. DOI: <https://doi.org/10.1016/j.matdes.2017.03.033>
- Xu, B., Wang, N., Chen, T., Li, M., May 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv e-prints*, arXiv:1505.00853.
- You, D., Gao, X., Katayama, S., 05 2014. Multisensor fusion system for monitoring high-power disk laser welding using support vector machine. *Industrial Informatics, IEEE Transactions on* 10, 1285–1295. DOI: 10.1109/TII.2014.2309482
- Zhang, Z., Chen, S., 10 2014. Real-time seam penetration identification in arc welding based on fusion of sound, voltage and spectrum signals. *Journal of Intelligent Manufacturing* 28, 1–12. DOI: 10.1007/s10845-014-0971-y
- Zhang, Z., Wen, G., Chen, S., 2019. Weld image deep learning-based on-line defects detection using convolutional neural networks for al alloy in robotic arc welding. *Journal of Manufacturing Processes* 45, 208 – 216. DOI: <https://doi.org/10.1016/j.jmapro.2019.06.023>
- Zhang, Z., Yu, H., Lv, N., Chen, S., 2013. Real-time defect detection in pulsed gtaw of al alloys through on-line spectroscopy. *Journal of Materials Processing Technology* 213 (7), 1146 – 1156. DOI: <https://doi.org/10.1016/j.jmatprotec.2013.01.012>

Appendix A. Overview of sound features extracted

| Feature name: | Symbol: | Matlab reference: |
|--------------------------------|----------------|--|
| Temporal: | | |
| Root amplitude | y_r | $\left(\frac{1}{N} \sum_{i=0}^{N-1} \sqrt{ y_i }\right)^2$ |
| Root mean square | y_{rms} | $rms(y)$ |
| Mean | \bar{y} | $mean(y)$ |
| Peak | \hat{y} | $max(y)$ |
| Shape factor | y_s | $\frac{y_{rms}}{\bar{y}}$ |
| Crest factor | y_c | $\frac{y}{y_{rms}}$ |
| Clearance factor | y_L | $\frac{y}{y_r}$ |
| Impulse factor | y_I | $\frac{y}{\bar{y}}$ |
| Median | y_{med} | $median(y)$ |
| Minimum | y_{min} | $min(y)$ |
| Mode | y_{mod} | $mode(y)$ |
| Variance | y_{σ^2} | $var(y)$ |
| Kurtosis | y_K | $kurtosis(y)$ |
| Skewness | y_S | $skewness(y)$ |
| Zero crossing rate | y_{ZCR} | $\frac{\sum_{i=1}^N diff(y_i > 0) }{N}$ |
| Peak to peak value | y_{p2p} | $peak2peak(y)$ |
| Peak to root mean square value | y_{p2rms} | $peak2rms(y)$ |
| Root sum squared | y_{rssq} | $rssq(y)$ |

Spectral shape:

| | | |
|--------------------------------------|------------|---|
| Spectral centroid | S_{sc} | $meanfreq(Pxx, F)$ |
| Spectral spread | S_{sp} | $\frac{\sum_{i=0}^{N-1} (F_i - S_{sc})^2 Pxx_i}{\sum_{i=0}^{N-1} Pxx_i}$ |
| Spectral skewness | S_S | $\frac{\sum_{i=0}^{N-1} (F_i - S_{sc})^3 Pxx_i}{\sum_{i=0}^{N-1} Pxx_i \cdot (S_{sc})^3}$ |
| Spectral kurtosis | S_K | $\frac{\sum_{i=0}^{N-1} (F_i - S_{sc})^4 Pxx_i}{\sum_{i=0}^{N-1} Pxx_i \cdot (S_{sc})^4}$ |
| Median frequency | S_{med} | $medfreq(Pxx, F)$ |
| Average power | S_{bw} | $bandpower(Pxx, F, psd')$ |
| Two-sided equivalent noise bandwidth | S_{enbw} | $enbw(y, Fs)$ |
| Occupied bandwidth | S_{obw} | $obw(Pxx, F)$ |
| 3 dB bandwidth | S_{pbw} | $powerbw(Pxx, f)$ |

Harmonic:

| | | |
|--------------------------------------|-------------|-----------------------|
| Total harmonic distortion | H_{thd} | $thd(Pxx, F, psd')$ |
| Signal to noise ratio | H_{snr} | $snr(Pxx, F, psd')$ |
| Signal to noise and distortion ratio | H_{sinad} | $sinad(Pxx, F, psd')$ |

Perceptual:

| | | |
|---------|------------|-----------------------------|
| 20 MFCC | P_{mfcc} | $mel_fcc(y, Fs, varargin)$ |
|---------|------------|-----------------------------|

Figure A.35: The complete set of features extracted from each window of sound (Bidstrup, 2017) (modified)

In the original thesis by Bidstrup, a set of a total 50 features were extracted from each time window, summarized in Figure A.35. (See (Bidstrup, 2017) p. 65-66). Furthermore the signal was decomposed using a Wavelet Packet Decomposition (WPD) tree, like the one illustrated on Figure A.36. At each node of the tree, the signal is split into two branches, each filtered through a low-pass and high-pass filter respectively, as shown on figure A.37. In order to reduce the amount of data accumulating at each division, the filtered signals are then downsampled by a factor of 2. The remaining datapoints are then called the *approximation coefficients* for the low-pass filtered data and *detail coefficients* for the high pass filtered (Michel Misiti, 1996). In Bidstrup's thesis decomposition tree of depth 5 is used. As the wavelet coefficients are also a data series, their properties can also be described through descriptive statistics. Therefore the 18 temporal features listed in Figure A.35 are also extracted at each of the tree nodes. This leads to a total of 1166 features being extracted for each analyzed time window:

$$\begin{aligned}
 N_{feat} &= n_{wpd} \cdot f_{temp} + f_{ss} + f_h + f_{prc} \\
 N_{feat} &= (2 \cdot 2^5 - 1) \cdot 18 + 9 + 3 + 20 \\
 &= 1166
 \end{aligned}$$

Where N_{feat} is the total number of features, n_{wpd} is the number of nodes on the WPD tree, and f_{temp} , f_{ss} , f_h and f_{prc} the number of temporal, spectral shape, harmonic and perceptual features listed in Figure A.35 respectively.

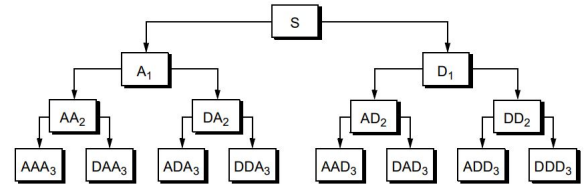


Figure A.36: A wavelet packet decomposition tree of depth 3. (Michel Misiti, 1996)

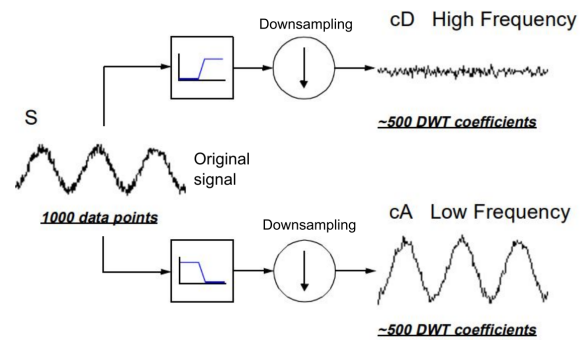


Figure A.37: A source signal S , being decomposed into a high-frequency set of detail coefficients cD and low-frequency approximation coefficients cA by the wavelet packet decomposition. This process is recursively repeated at every node of the WPD tree. (Michel Misiti, 1996)

Appendix B. Additional figures from PCA analysis of dataset

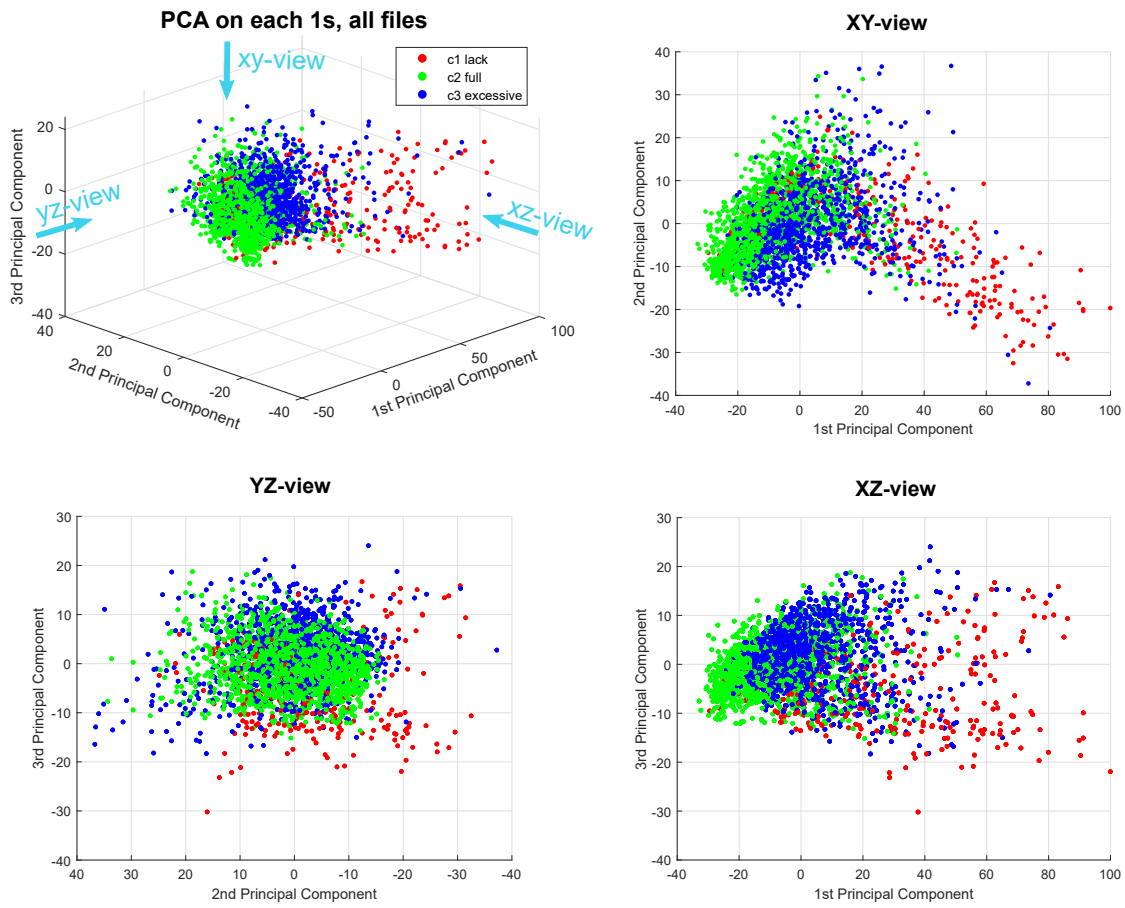


Figure B.38: 3D plot of the 3 first Principal components of the 1166 sound features extracted for each second of sound in the dataset. The first three components explain 45% of features variance.

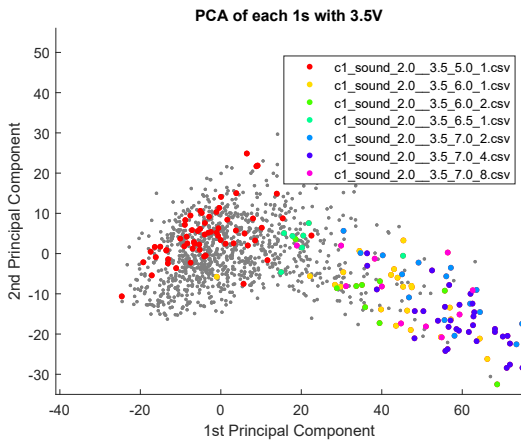


Figure B.39: Principal Components for the sound features of each second of data in the v35 set, c_1 files highlighted and grouped by filename. The file *c1_sound.2.0.3.5.5.0.1.csv* is treated as an outlier and excluded from the v35 set.

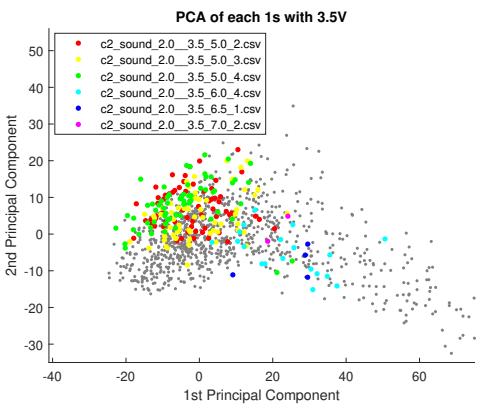


Figure B.40: Principal Components for the sound features of each second of data in the v35 set, c_2 files highlighted and grouped by filename.

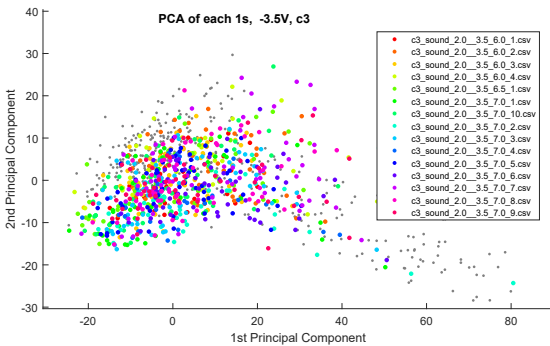


Figure B.41: Principal Components for the sound features of each second of data in the v35 set, c_3 files highlighted and grouped by filename.

Appendix C. Baseline architecture candidates

| 01 Arch | | | | | |
|---------|---------|--------------------|--------|-----------------|---------------------|
| Layer | Type | size (h,w,maps) | filter | stride (v,h) | trainable params |
| In | Image | 129x173x8 | - | - | - |
| C1 | Conv | 65x173x8 | 3x3 | 2,1 | 80 |
| P2 | MaxP | 32x86x8 | 2x2 | 2,2 | - |
| C3 | Conv | 16x86x16 | 3x3 | 2,1 | 1.168 |
| P4 | MaxP | 8x43x16 | 2x2 | 2,2 | - |
| FL5 | Flatten | 1x5504x1 | - | - | - |
| D6 | Dense | 1x96x1 | - | - | 528.480 |
| Out | Dense | 1x3x1 | - | - | 291 |
| sum: | | | | | 530.019 |

Table C.9

| 04 Arch | | | | | |
|---------|---------|--------------------|--------|-----------------|---------------------|
| Layer | Type | size (h,w,maps) | filter | stride (v,h) | trainable params |
| In | Image | 129x173x1 | - | - | - |
| C1 | Conv | 129x173x8 | 3x3 | 1,1 | 80 |
| C2 | Conv | 129x173x8 | 2x2 | 1,1 | 584 |
| P3 | MaxP | 64x86x8 | 3x3 | 2,2 | - |
| C4 | Conv | 62x84x16 | 2x2 | 1,1 | 1.168 |
| C5 | Conv | 60x82x16 | 3x3 | 1,1 | 2.320 |
| P6 | MaxP | 30x41x16 | 2x2 | 2,2 | - |
| FL7 | Flatten | 19680 | - | - | - |
| D8 | Dense | 1x512x1 | - | - | 10.076.672 |
| Out | Dense | 1x3x1 | - | - | 1.539 |
| sum: | | | | | 10.082.363 |

Table C.12

| 02 Arch | | | | | |
|---------|---------|--------------------|--------|-----------------|---------------------|
| Layer | Type | size (h,w,maps) | filter | stride (v,h) | trainable params |
| In | Image | 129x173x1 | - | - | - |
| C1 | Conv | 129x173x2 | 3x3 | 1,1 | 80 |
| P2 | MaxP | 64x86x8 | 2x2 | 2,2 | - |
| C3 | Conv | 62x84x16 | 3x3 | 1,1 | 1.168 |
| P4 | MaxP | 31x42x16 | 2x2 | 2,2 | - |
| FL5 | Flatten | 1x20832x1 | - | - | - |
| D6 | Dense | 1x120x1 | - | - | 2.499.960 |
| D7 | Dense | 1x84x1 | - | - | 10.164 |
| Out | Dense | 1x3x1 | - | - | 255 |
| sum: | | | | | 2.511.627 |

Table C.10

| 03 Arch | | | | | |
|---------|---------|--------------------|--------|-----------------|---------------------|
| Layer | Type | size (h,w,maps) | filter | stride (v,h) | trainable params |
| In | Image | 129x173x1 | - | - | - |
| C1 | Conv | 127x171x32 | 3x3 | 1,1 | 320 |
| P2 | MaxP | 63x85x32 | 2x2 | 2,2 | - |
| C3 | Conv | 61x83x64 | 3x3 | 1,1 | 18.496 |
| P4 | MaxP | 30x41x64 | 2x2 | 2,2 | - |
| C5 | Conv | 28x39x128 | 3x3 | - | 73.856 |
| P6 | MaxP | 14x19x128 | 2x2 | - | - |
| FL7 | Flatten | 1x34048x1 | - | - | - |
| D8 | Dense | 1x64x1 | - | - | 2.179.136 |
| Out | Dense | 1x3x1 | - | - | 195 |
| sum: | | | | | 2.272.003 |

Table C.11