

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
PROGRAMA DE PÓS-GRADUAÇÃO EM AUTOMAÇÃO E  
SISTEMAS**

Denise Albertazzi Gonçalves

**A CONVOLUTIONAL NEURAL NETWORK APPROACH ON  
BEAD GEOMETRY ESTIMATION FOR A LASER CLADDING  
SYSTEM**

Dissertação submetido(a) ao Programa  
de Pós-Graduação em Engenharia de  
Automação e Sistemas da  
Universidade Federal de Santa  
Catarina para a obtenção do Grau de  
Mestre em Engenharia de Automação e  
Sistemas  
Orientador: Prof. Dr. Marcelo Ricardo  
Stemmer

Florianópolis  
2019

Ficha de identificação da obra elaborada pelo autor  
através do Programa de Geração Automática da Biblioteca Universitária  
da UFSC.

Gonçalves, Denise Albertazzi  
A CONVOLUTIONAL NEURAL NETWORK APPROACH ON BEAD  
GEOMETRY ESTIMATION FOR A LASER CLADDING SYSTEM /  
Denise Albertazzi Gonçalves ; orientador, Marcelo  
Ricardo Stemmer, 2019.  
86 p.

Dissertação (mestrado) - Universidade Federal de  
Santa Catarina, Centro Tecnológico, Programa de Pós  
Graduação em Engenharia de Automação e Sistemas,  
Florianópolis, 2019.

Inclui referências.

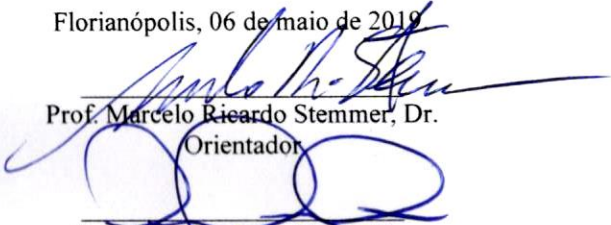
1. Engenharia de Automação e Sistemas. 2. Redes  
Neurais Convolucionais. 3. Laser Cladding. 4.  
Estimação de geometria. I. Stemmer, Marcelo Ricardo.  
II. Universidade Federal de Santa Catarina.  
Programa de Pós-Graduação em Engenharia de Automação e  
Sistemas. III. Título.

# A CONVOLUTIONAL NEURAL NETWORK APPROACH ON BEAD GEOMETRY ESTIMATION FOR A LASER CLADDING SYSTEM

Denise Albertazzi Gonçalves

Esta Dissertação/Tese foi julgada adequada para obtenção do Título de  
"Mestre" e aprovada em sua forma final pelo Programa de Pós-  
Graduação em Engenharia de Automação e Sistemas

Florianópolis, 06 de maio de 2019

  
Prof. Marcelo Ricardo Stemmer, Dr.

Orientador

Prof. Werner Kraus Júnior, Dr.

Coordenador do Programa de Pós-Graduação em Automação e Sistemas

## Banca Examinadora:

  
Prof. Marcelo Ricardo Stemmer, Dr.

Presidente

Universidade Federal de Santa Catarina

  
Prof. Milton Pereira, Dr.

Universidade Federal de Santa Catarina

  
Prof. Walter Lindolfo Weingaertner, Dr.

Universidade Federal de Santa Catarina

  
Prof. Maurício Edgar Stivanello, Dr.

Instituto Federal de Santa Catarina

**Prof. Jomi Fred Hübner, Dr.**  
DAS - PPGEAS/UFSC - CTC



This work is dedicated to my family  
and to my laboratory coworkers.



## ACKNOWLEDGEMENTS

The author would like to thank the *Laboratório de Mecânica de Precisão* (Precision Mechanics Laboratory) staff for making this work possible. She would also like to thank the Labmetro for the hardware support.

The author would also like to thank her family for support during those long two years.

The author would like to thank the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* CNPq for funding this work.





## RESUMO EXTENDIDO

### Introdução

Laser cladding é um processo de fabricação baseado em manufatura aditiva no qual um laser é utilizado para fundir material de adição sobre um substrato, sobre o qual uma poça fundida é formada e, desta, forma-se um cordão de cladding. Dentre suas vantagens estão o baixo aporte térmico e elevada qualidade superficial de suas peças produzidas. Tal processo é, porém, altamente susceptível a perturbações, resultando em alterações da geometria final de seus produtos. O Laboratório de Mecânica de Precisão (LMP) na Universidade Federal de Santa Catarina (UFSC) possui um sistema laser de alta potência capaz de operar diversos processos de manufatura a laser, incluindo laser cladding. Neste sistema, busca-se compreender a influência dos parâmetros de processo sobre a geometria final dos cordões de cladding produzidos.

### Objetivos

Propõe-se um método automático para a estimação da geometria final do cordão para um processo de laser cladding. Como objetivos secundários encontram-se a adequação do sistema para a aquisição das imagens, a medição real da geometria dos cordões de cladding e o desenvolvimento de diferentes arquiteturas de redes neurais convolucionais.

### Metodologia

Primeiramente, são realizadas revisões bibliográficas sobre os principais conceitos das áreas de laser cladding e de redes neurais convolucionais. Em seguida, buscam-se aplicações de ambas as técnicas combinadas, novamente na literatura, de forma sistemática. Inicia-se, então, a adaptação do sistema de laser cladding para a aquisição das imagens da poça fundida. Os cordões de cladding são então fabricados e, posteriormente, medidos através de fotogrametria ativa. Finalmente, as redes neurais convolucionais são desenvolvidas na linguagem Python 3.6 utilizando-se da biblioteca Keras.

### Resultados e Discussão

As redes neurais convolucionais desenvolvidas são capazes de estimar a geometria final dos cordões com alta acurácia. O coeficiente de determinação entre os valores reais e estimados pelas redes ultrapassa 0.95 para cada frame nos melhores casos. O erro médio, considerando todos os cordões, chega a valores tão reduzidos como 5  $\mu\text{m}$ . Algumas

arquiteturas são mais susceptíveis ao fenômeno de overfitting que outras, embora este fenômeno não seja suficiente para invalidar seus resultados.

### **Considerações Finais**

Como uma primeira abordagem de monitoramento ótico inteligente no laboratório, os resultados foram muito positivos. O sistema laser foi adaptado para adquirir imagens do processo, enquanto as redes foram capazes de estimar a geometria final dos cordões de cladding com sucesso. Com este sistema, o laboratório está mais próximo da implementação de um futuro controle em malha fechada do processo.

**Palavras-chave: Redes Neurais Convolucionais, CNN, laser cladding, geometria de cordão, estimativa de geometria, monitoramento ótico.**

## ABSTRACT

Laser cladding is a complex manufacturing process which requires fine-tuning to achieve the desired geometry. In order to further understand the process, an automated method for clad bead final geometry estimation on a laser cladding system is proposed. To do so, convolutional neural network architectures were developed. They receive the camera image and process parameters as inputs, yielding width and height of the clad beads as outputs. The optical monitoring system's hardware was updated as well. The results of the network's performances show coefficients of determination between the target and the estimated values above 0.95 for each frame on the best cases, while the error mean among all clad beads get to as little as 5  $\mu\text{m}$ . Those results take the laboratory one step further into closed loop control for this process.

**Keywords:** Convolutional neural network, CNN, laser cladding, bead geometry, geometry estimation, optical monitoring.



## IMAGE INDEX

|   |    |
|---|----|
| Figure 1 - Laser Cladding Setup (a) and optical path schematic (b). ....  | 26 |
| Figure 2 - Molten pool image, 1750W, 800mm/min. On this image, the cladding head is traveling to the right relative to the substrate, parallel to it. ....                      | 27 |
| Figure 3 - Clad bead manufacturing schematic. ....  | 28 |
| Figure 4 - Convolution operation. ....  | 30 |
| Figure 5 - Pooling operation. ....  | 31 |
| Figure 6 - Search queries for optical system monitoring (left) and artificial intelligence for laser cladding (right). ....   | 35 |
| Figure 7 - System with two cameras, one on a shallow angle and the other on a 45° angle. ....   | 36 |
| Figure 8 - Molten pool images: (a) high-speed camera image; (b) infrared coaxial camera image; (c) molten pool isotherm. ....   | 38 |
| Figure 9 - Schematic of the trinocular CCD based detection system. ...  | 40 |
| Figure 10- Experimental setup with a webcam and UV illumination. ..   | 41 |
| Figure 11 - Experimental average and ANN predictions for depth. ....  | 44 |
| Figure 12 - Scatter diagram with the best fit of GA-BPNN prediction vs. experimental. (a) training patterns (b) test patterns. ....   | 45 |
| Figure 13 - Analog camera, model CF 8/5 MX from Kappa (a). Image converter, model Video-to-USB 2.0 converter from The ImageSource (b). ....                                     | 48 |
| Figure 14 - Digitalized image from the analog camera. ....  | 48 |
| Figure 15 – Designed filter support. It fits up to two 25.4 mm (1 inch) diameter filters. ....  | 49 |
| Figure 16 - Clad bead layout. ....  | 50 |
| Figure 17 - Cladbed beads. Clads deposited on red rectangles correspond to the three sets of data used in this work. Marks on the blue rectangle were made without powder. .... | 51 |
| Figure 18 - Clad bead height and width geometries. ....   | 52 |
| Figure 19 - Clad beads data cloud. ....   | 53 |
| Figure 20 - Clad beads with sectioning. ....  | 54 |
| Figure 21 - Image from training set before preprocessing. Nozzle area and cropped image regions. 1050W, 300mm/min. 1200 x 1600 pixels. ....                                     | 55 |
| Figure 22 -Figure 21 after cropping, masking and rescaling. 128 x 128 pixels. ....  | 55 |
| Figure 23 - Image generator code. ....  | 57 |
| Figure 24 - Schematic for CNN architecture. ....  | 59 |
| Figure 25 – Image branch code. ....   | 60 |

|   |    |
|---|----|
| Figure 26 - Input parameters branch code.....   | 61 |
| Figure 27 - Merging branch code.....  | 61 |
| Figure 28 - CNN loss, optimizer, and metrics. Source: Author. ....  | 62 |
| Figure 29 - Convolutional neural network prediction gathering code. .   | 63 |
| Figure 30 - Error mean and standard deviation calculation. ....   | 64 |
| Figure 31 – Loss value from 200 epochs training of all neural networks with both width and height outputs.....  | 66 |
| Figure 32 – Loss value from 200 epochs training of all neural networks with width as output.....  | 66 |
| Figure 33 – Loss value from 200 epochs training of all neural networks with only height as output.....  | 67 |
| Figure 34 – Mean absolute error values from 200 epochs training of all neural networks with both width and height as outputs, in millimeters.   | 68 |
| Figure 35 – Mean absolute error values from 200 epochs training of all neural networks with width as output, in millimeters. ....   | 68 |
| Figure 36 – Mean absolute error values from 200 epochs training of all neural networks with height as output, in millimeters. ....  | 69 |
| Figure 37 – Percentage error values from 200 epochs training of all neural networks with both width and height as outputs.....  | 70 |
| Figure 38 – Percentage error values from 200 epochs training of all neural networks with width as output. ....  | 70 |
| Figure 39 – Percentage error values from 200 epochs training of all neural networks with height as output. ....   | 71 |
| Figure 40 – Best and worst network performances from both testing and training phases. Top row: Training phase. Bottom row: Testing phase. Left column: Best performance. Right column: Worst performance.... | 72 |
| Figure 41 - $R^2$ for the multiple output networks, calculated for both outputs (top), only for width (bottom left) and only for height (bottom right).....   | 75 |
| Figure 42 – Gaussian distributions of the estimation error for the networks with multiple outputs on their training phase (left) and testing phase (right). ....  | 77 |
| Figure 43 – Gaussian distributions of the estimation error for the networks with the width output on their training phase (left) and testing phase (right). ....  | 78 |
| Figure 44 – Gaussian distributions of the estimation error for the networks with the height output on their training phase (left) and testing phase (right). ....   | 79 |

## TABLE INDEX

|   |    |
|---|----|
| Table 1 - Input process parameters. Source: Author .....  | 50 |
| Table 2 - Theoretical number of frames for each travel speed.....   | 53 |
| Table 3 - Mean squared error and mean absolute error functions.....   | 62 |
| Table 4 – Convolutional Neural Network Architectures.....   | 62 |
| Table 5 – R <sup>2</sup> value for the networks with both width and height outputs.<br>.....  | 73 |
| Table 6 – R <sup>2</sup> value for the networks with width output.....  | 74 |
| Table 7 – R <sup>2</sup> value for the networks with height output.....   | 74 |
| Table 8 – Individual height and width R <sup>2</sup> for all networks.....  | 76 |
| Table 9 – Mean and standard deviations for the estimation error on both<br>training and testing phases for the multiple output networks.....    | 77 |
| Table 10 – Mean and standard deviations for the estimation error on both<br>training and testing phases for the width output networks. ....     | 78 |
| Table 11 – Mean and standard deviations for the estimation error on both<br>training and testing phases for the height output networks. ....    | 79 |
| Table 12 – Error mean and standard deviation of the test phases from<br>networks A and F for all output types. ....                             | 80 |
| Table 13 - Percentual error definition. ....  | 80 |
| Table 14 – Percentual error mean and percentual error standard deviation<br>of the test phases from networks A and F for all output types. .... | 81 |





## **LIST OF ABBREVIATIONS AND ACRONYMS**

ANN – Artificial Neural Network  
CAD – Computer Aided Design  
CAM – Computer Aided Manufacturing  
CCD – Charged Coupled Device  
CMOS – Complementary Metal-Oxide Semiconductor  
CNN – Convolutional Neural Network  
CSV – Comma Separated Values  
FDM – Fused Deposition Modeling  
FPGA – Field Programmable Gate Array  
fps – frames per second  
GA – Genetic Algorithm  
GA-BPNN – Genetic Algorithm based Backpropagation Neural Network  
IR – Infrared  
LMP – Laboratório de Mecânica de Precisão (Precision Mechanics Laboratory)  
MAE – Mean Absolute Error  
MIG – Metal Inert Gas  
MSE – Mean Squared Error  
MTU – Maximum Transmit Unit  
PI – Proportional Integral  
PID – Proportional Integral Derivative  
RNN – Recurrent Neural Network  
UFSC – Universidade Federal de Santa Catarina (Federal University of Santa Catarina)  
UV – Ultraviolet



## INDEX

|              |  |           |
|--------------|--|-----------|
| <b>1</b>     | <b>Introduction.....</b>                             | <b>21</b> |
| 1.1          | Objectives.....                                      | 22        |
| 1.2          | Contributions.....                                   | 22        |
| 1.3          | Text Structure.....                                  | 22        |
| <b>2</b>     | <b>Theoretical background.....</b>                   | <b>25</b> |
| 2.1          | Laser Cladding .....                                 | 25        |
| <b>2.1.1</b> | <b>Laser Cladding Setup .....</b>                    | <b>25</b> |
| <b>2.1.2</b> | <b>Molten pool .....</b>                             | <b>26</b> |
| <b>2.1.3</b> | <b>Clad Bead.....</b>                                | <b>27</b> |
| <b>2.1.4</b> | <b>Process Parameters .....</b>                      | <b>28</b> |
| 2.2          | Convolutional Neural Networks .....                  | 29        |
| <b>2.2.1</b> | <b>Convolution Layer .....</b>                       | <b>30</b> |
| <b>2.2.2</b> | <b>Pooling Layer.....</b>                            | <b>31</b> |
| <b>2.2.3</b> | <b>Dense Layers and Output.....</b>                  | <b>31</b> |
| <b>2.2.4</b> | <b>Learning.....</b>                                 | <b>32</b> |
| <b>3</b>     | <b>State of the Art .....</b>                        | <b>35</b> |
| <b>4</b>     | <b>Implementation .....</b>                          | <b>47</b> |
| 4.1          | Optical system adequation .....                      | 47        |
| 4.2          | Laser clad processing .....                          | 50        |
| 4.3          | Clad bead geometry measurement .....                 | 52        |
| 4.4          | Image preprocessing .....                            | 54        |
| 4.5          | Neural network development and evaluation.....       | 56        |
| <b>4.5.1</b> | <b>Image Generator .....</b>                         | <b>56</b> |
| <b>4.5.2</b> | <b>Convolutional Neural Network Layers .....</b>     | <b>58</b> |
| <b>4.5.3</b> | <b>Convolutional Neural Network Evaluation .....</b> | <b>63</b> |
| <b>5</b>     | <b>Results .....</b>                                 | <b>65</b> |
| 5.1          | Training results .....                               | 65        |
| 5.2          | Estimation performance .....                         | 71        |

|              |   |           |
|--------------|---|-----------|
| <b>5.2.1</b> | <b>Target and Prediction values .....</b> | <b>72</b> |
| <b>5.2.2</b> | <b>Estimation error.....</b>              | <b>76</b> |
| <b>6</b>     | <b>Conclusion .....</b>                   | <b>83</b> |
| <b>7</b>     | <b>Bibliography.....</b>                  | <b>87</b> |

## 1 Introduction

Laser cladding is an additive manufacturing process in which a laser beam melts feedstock material into a substrate, producing a clad bead. On the last decades, its usage has increased due to minimal thermal distortion, minimal dilution, and high superficial quality (TOYSERKANI, 2005). It is a highly multidisciplinary process, embracing subareas such as laser technology, numeric command, powder metallurgy, process control, and monitoring (TOYSERKANI, 2005). Coating, wear off parts repair, fabrication of parts with variable mechanical properties, and complex geometries are all suitable applications for this process.

Together with this process' boom came the need for its monitoring. Having such a complex physical nature, the understanding of this process is of high interest for its users. To identify the different aspects which influence a final part's geometry is to enhance its efficiency, achieving a final geometry closer to the target one, thus, leading to less post-processing time.

The *Laboratório de Mecânica de Precisão* (Precision Mechanics Laboratory – LMP) on the *Universidade Federal de Santa Catarina* (Federal University of Santa Catarina – UFSC) possesses two laser processing units in which processes such as laser cladding, laser remelting, laser autogenous welding, laser-MIG hybrid welding, and laser superficial treatment are studied. One of the units utilizes a 10 kW laser source, in which most of those processes occur. In this unit's setup, there is a camera which acquires images coaxially to the laser beam. Through this camera it is possible to acquire images of the molten pool – the region where the laser melts the feedstock material together with the substrate, forming the clad bead. This molten pool holds most of this process' secrets, providing the most information out of it.

An efficient way to gather information from the molten pool is to acquire its image. Using such a camera setup as described previously, one can acquire images of it which are normal to the substrate. Those images provide information on molten pool geometry and on its brightness. When properly analyzed, those images have the potential to yield information related to the clad bead's geometry.

Inspired by those who run the laser unit, a monitoring system, and its software are here proposed. It aims the understanding of the process just as operators do. A glance at the molten pool's image during the process is all that it is needed for an experienced operator to identify an overly heated molten pool, which they easily correct by adjusting laser power. If

the human neural network can perform it, this is certainly an inspiration for artificial networks to do so.

Taking a step further, not only could a neural network identify an over – or under – heated molten pool, but, perhaps, the very geometry of the clad bead. As the superficial tension of the molten pool drops with temperature, so does the height of the bead – as the material flows further over the substrate, increasing bead width.

For such a daring task, the chosen neural network architecture once more follows the human sight. Inspired by the vertebrae cortex, the convolutional neural network is the perfect match for this challenge, nevertheless, this combination of laser cladding and convolutional neural networks is yet to be found in the literature.

## 1.1 Objectives

The main objective of this work consists of the following:

“To develop an automated method for clad bead final geometry estimation on a laser cladding system”

In order to do so, some steps were identified as essential to this work, which are here named as secondary objectives:

- To analyze the current optical system and, if necessary, propose suitable modifications to it
- To develop a methodology for the reliable measurement of the final clad bead geometry
- To develop different convolutional neural network architectures for clad bead geometry estimation

## 1.2 Contributions

The execution of this work results in the first assault towards intelligent process monitoring for the laboratory. It opens research lines, which enables future works to be executed in these fields. Besides that, it leaves the laboratory one step further to process control, as process monitoring is the key for such. For the scientific community, this works contributes with yet another possibility of process monitoring, leaving room for many optimizations.

## 1.3 Text Structure

The first step of this work was to research literature on laser cladding and convolutional neural networks for their fundamentals, in chapter 2. Later, another literature research was performed, this time looking for the efforts made on laser cladding process optical monitoring and on neural network applications for clad bead estimation since the beginning of the technology, on chapter 3.

After the research was concluded, the optical system adequation which would allow image acquisition initiated. The usage of different lenses, filters, and cameras was analyzed, and a final setup was implemented. Then, the clad beads were deposited, their images, acquired. The beads were measured through active photogrammetry and the images were preprocessed on the Python 3.6 language using the OpenCV library. Six different convolutional neural network architectures were then developed under the Python 3.6 language by using the Keras library with the TensorFlow backend. Metrics were also developed aiming to enhance the architecture's comparability. Other libraries such as Numpy, Scipy, Pandas, and Glob were also used for support operations and file management. These development and implementation steps are in chapter 4.

The results are presented in chapter 5. There, the architecture's performances are compared. Chapter 6 concludes this work, analyzing its successes and failures, as well as presents suggestions for those who dare to follow the path this work initiated. On chapter 7, the references used throughout this work as listed.





## **2 Theoretical background**

This chapter is designed to explain the major areas of this work. It is performed with a standard, exploratory research on the most consolidated literature on the area, along with community used reference websites. The first section details how the laser cladding process works and its most important features, while the last section explains how a convolutional neural network is structured and how it operates on images.

### **2.1 Laser Cladding**

Even naming this technology has proven to be a complex task. Laser cladding, laser metal deposition, laser coating, laser powder deposition, laser surfacing, laser direct deposition or even laser additive manufacturing are all usual names for such a process. Laser cladding is an interdisciplinary laser-based technology, which can also embrace CAD/CAM software, robotics, sensors and control, and powder metallurgy technologies (TOYSERKANI, 2005).

The growth of laser cladding technology increases among other manufacturing technologies due to its advantages, such as enhanced thermal control, reduction of production time and unique, smart structures production (TOYSERKANI, 2005).

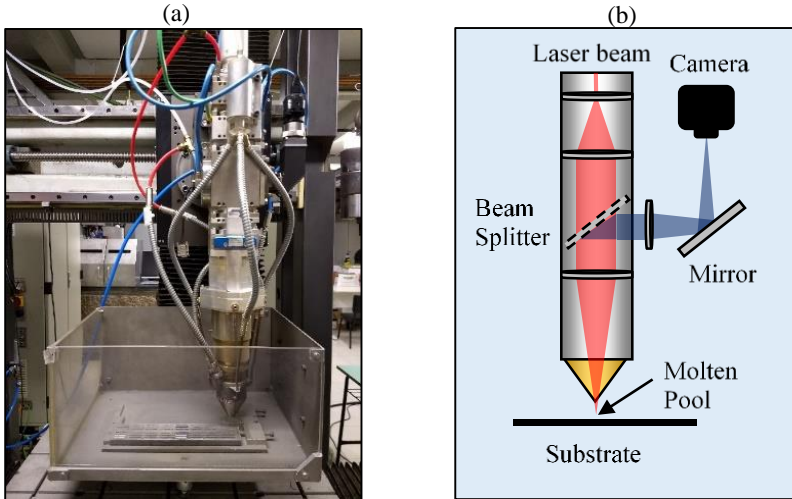
This section explains how laser cladding works as a plant to be controlled, which means, with a black box point of view. No phenomenological details are studied as they are not the focus of this work. As so, a quick overview of the process setup is presented. Subsequently, the molten pool is defined and detailed. After that, the main input and output variables are listed. Finally, the final product – the clad bead – is presented and detailed.

#### **2.1.1 Laser Cladding Setup**

Laser cladding is a laser-based additive manufacturing process where a laser beam is used to melt feedstock material over a substrate to build structures. The purpose of those structures can range from a surface coating to complex geometric build-up, being possible to select their chemical composition throughout a broad spectrum. There are many options or possibilities regarding feedstock material feeding setup, being the powder coaxial feeding method the one of interest in this work. In this method, the laser beam travels inside the laser cladding head, while

powder is blown coaxially to it from a ring-shaped aperture. An inert gas acts as a powder carrier gas, protecting it against oxidation. The laser cladding head in Figure 1a illustrates such setup.

Figure 1 - Laser Cladding Setup (a) and optical path schematic (b).

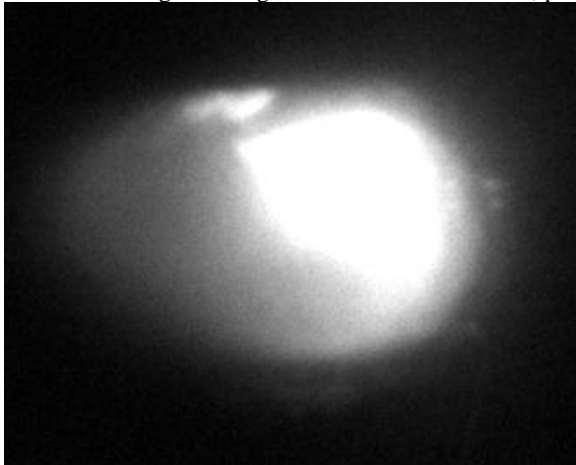


Included in this system there is a camera set coaxially to the laser beam, which allows visualization of the molten pool – the region where the laser melts powder and substrate, further detailed on section 2.1.2. For acquiring molten pool images, firstly, its radiation and laser reflections travel into the cladding head. While the laser wavelength ( $1070\ \mu\text{m}$ ) passes straight through the beam splitter – a half mirror – visible and shorter IR wavelengths are reflected by it. Without this beam splitter filtering out the laser radiation, any camera would suffer damage from it. Another mirror directs the shorter wavelengths to the camera. This optical path is illustrated in Figure 1b.

### 2.1.2 Molten pool

During the process, the laser beam melts powder and substrate, forming the molten pool. A strong metallurgical bond is then formed. This is crucial for adhering the newly deposited layer over the substrate. An image of a molten pool can be seen in Figure 2.

Figure 2 - Molten pool image, 1750W, 800mm/min. On this image, the cladding head is traveling to the right relative to the substrate, parallel to it.

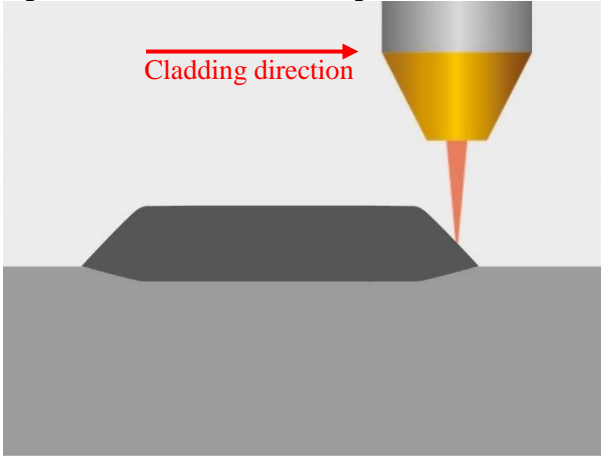


The molten pool image brings information on the resulting structure. The brighter the image, the hotter the molten pool is. The temperature can relate to the resulting microstructure, leading to different mechanical properties. Its geometry is also relevant for the resulting clad bead geometry, as explained in section 2.1.3.

### **2.1.3 Clad Bead**

A clad bead is the simplest product from a laser cladding process. It is constructed by moving the cladding head over the substrate on a cladding direction while keeping the laser beam on and feeding powder. After molten pool solidification, the resulting structure is a bead made of a mixture of substrate and powder materials. A schematic of such a manufacturing process can be seen in Figure 3.

Figure 3 - Clad bead manufacturing schematic.



On additive manufacture, a structure is built in layers. Clad bead height is directly related to layer height, which usually diminishes on the later layers due to thermal build up. A good estimation of clad bead height allows a better in process control. Also on this process, clad bead width influences in surface quality. To maintain the same bead width throughout a process with considerable thermal buildups, which are usual, is to considerably improve the surface quality and layer consistency. Both dimensions are also important in coating processes.

There are other dimensions or features relevant to the clad geometry on the literature, such as penetration depth, bead area, and dilution. Those dimensions are not the focus on this work, thus, they will not be defined here.

Nonetheless, to achieve any desired geometry, process tuning is mandatory. Its input parameters need to be comprehended. A highlight of its most important parameters can be seen in the next section, 2.1.4.

#### 2.1.4 Process Parameters

For achieving different clad bead geometries, process parameters need to be altered accordingly. Laser power, travel speed, powder flow, laser focus spot size, and laser beam energy density are just some examples. On this work, the foci are the laser power and travel speed parameters.

Each of those parameters has a different influence on the clad bead. More laser power leads to higher temperatures, which increases molten pools wettability, thus, reducing clad bead height and increasing clad bead width. Faster travel speeds reduce the quantity of deposited material on the clad bead, shrinking all dimensions.

The influence of process parameters on clad bead dimensions is quite clear. What links them both is the molten pool. Acquired by a coaxial camera, its imaging can provide insightful information regarding the process and its outcomes. An efficient way to extract this information is through artificial intelligence, with the use of convolutional neural networks.

This specific type of neural network fits this problem because of its property of having the raw image as an input, with no need for further image preprocessing. There is also no need for defining any descriptors for the molten pool, e.g area, width, nor length. The CNN is explained in the next section, 2.2.

## 2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are inspired by the vertebrate's visual cortex. It is a hierarchical architecture focused on image and audio analysis in which lower layers identify simple patterns, outputting feature maps. Those maps are then fed to higher layers, which identify more complex patterns, and so on.

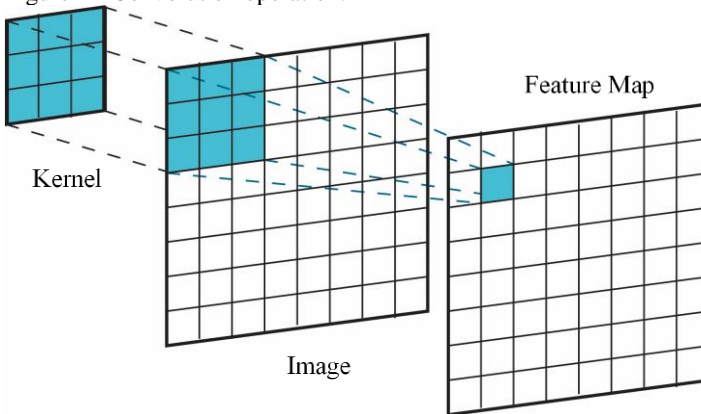
Although fairly used in many fields to this day, the creation of the CNN basics is almost four decades old. With the name of s-layers and c-layers on the "Neocognitron", the convolution and pooling layers were firstly drafted (FUKUSHIMA, 1980). As the research continued, a receptive field – which today is known as a kernel – was used to scan the image, achieving pattern shift invariance (LE CUN, 1989). The main architecture of CNN was then created.

A CNN is composed of two main types of layers: convolution and pooling layers, which always comes in pairs. The following sections, 2.2.1 and 2.2.2, explain each of those layers. A CNN can have one or many convolution-pooling layer pairs, but after those, there are always densely connected layers. Those layers outputs can be either a classification or a regression, as discussed in section 2.2.3. Finally, the CNN learning mechanism is briefly explained in section 2.2.4.

### 2.2.1 Convolution Layer

As the name may suggest, convolution layers are the heart of CNN. They perform the convolution operation across an image. This operation consists of multiplying each element of the kernel – a small matrix with predetermined values – times the pixels in a region of the same size in the input image. Then, the values from the resulting matrix are added and stored in an equivalent position on a feature map. This algorithm is illustrated in Figure 4. Later, the kernel is shifted to an adjacent position – defined by a stride value, which is the number of cells to shift – where the algorithm is repeated, and a new value is stored in an adjacent position in the feature map. The output is the feature map matrix (INC., 2018)

Figure 4 - Convolution operation.



One can think that this operation alters the image size. There are options for it not to happen. The padding type defines it, where zero values are annexed on the sides of the image so that the kernel multiplication can be performed. Another viable padding option is to discard the edge values.

This operation can be performed with one or many kernels. Each kernel then detects a different feature on the image, e.g. straight edges, shapes, etc. With multiple kernels, the resulting feature maps gain an additional dimension. The resulting output has four dimensions: height, width, color space and feature maps.

It is a good practice to use normalized data. As neural networks are a matrix multiplication based algorithm, non-normalized data can induce mathematical complications to it. Batch normalization layers are usually

added right after the convolution layers for batch-wise data normalization. Batches are sets of images fed together to the network, improving training performance, but increasing memory usage. The batch normalization layers subtract the batch mean value from all the images, and divide them by their standard deviation (DOUKKALI, 2017). This operation is also performed on the input data.

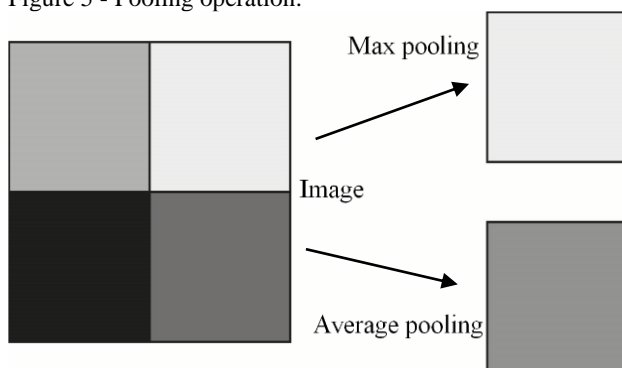
The next step is to feed this normalized result in the pooling layer.

### 2.2.2 Pooling Layer

After the convolution operation, the feature maps are fed into the pooling layer. What happens in this layer is a simple undersampling. The feature maps are then scaled down on their height and width dimensions but kept unchanged on their color space and feature maps dimensions. This allows a next convolutional layer to convolve on a larger area on the original image. Some details are lost with the increase in area, but those details have already been processed by the previous convolutional layer.

There are many ways to perform the pooling operation. Two usual ways are max pooling and average pooling. In both ways, an area is replaced by a single value that represents it. On the average pooling method, the average pixel value from the feature map is chosen to replace the whole area. On the max pooling method, the max value of the area replaces it, signifying that a feature was detected. Both cases of the pooling operation are depicted in Figure 5.

Figure 5 - Pooling operation.



### 2.2.3 Dense Layers and Output

After the convolutional and pooling layers, the feature maps enter a densely connected network. This type of network has connections from every neuron on one layer to every neuron on the next layer. Each connection has a weight, which is a value that is multiplied by the input. The weights give more significance to some inputs than to others, although the weight values differ for every neuron.

To connect the CNN architecture to a densely connected network, the data needs to be flattened, turning into a single column array. This data array is then fed to the dense layers, which calculate the output of the CNN.

There are mainly two types of operations a CNN can perform: classification and regression. The classification operation returns a class in which the image may belong, while the regression operation returns a value instead. How these outputs are calculated is explained in section 2.2.4.

## **2.2.4 Learning**

CNN is a supervised learning algorithm, meaning it needs the expected network output in order to learn. A training phase is required for both classification and regression operations. Thus, it is necessary to split the available data into at least two sets correspondent to the two learning phases: training and testing.

The training phase's objective is to minimize the error between the network outputs and their target values by changing the weights between neuron connections. The training starts with the network having random weight values, thus random output values. A function which compares this estimated values to the real ones is a loss function. To minimize this function's output – named the loss value – is the aim of the training phase.

The CNN weights are adjusted in order to minimize the loss value. This adjustment is made by an optimizer algorithm, responsible for finding a local minimum for the loss function. Each weight adjustment and loss value recalculation have the name epoch. A CNN training ends after a predetermined number of epochs.

For the sake of visualization, metrics can be used. Those do not interfere with the training itself but serve as a way for the developer to further understand how well is the network training.

After this, starts the testing phase. The CNN then calculates the output from new input. Both results are compared, and the loss value is calculated, along with any other desired metrics. For classification



problems, it is usual to build a matrix with the predicted and actual results – one on each axis – named confusion matrix. For regression, however, it is more usual to make a graph with the predicted and actual results – also one on each axis –, calculate the best fit line and its  $R^2$  factor.

Once understood the main concepts on the laser cladding and neural network areas, what has been done when one combines both technologies can be discussed. The state of the art for neural network usage on laser cladded bead geometry estimation, as well as the first image based monitoring systems for such a process, can be acknowledged in chapter 3.



### 3 State of the Art

Laser cladding process monitoring has been a field of research for at least two decades. During this period, many systems and strategies were developed to better understand this process, thus improve its quality. The papers here presented include research on the image acquisition system, molten pool segmentation and temperature, bead and molten pool geometries, as well as machine learning-based approaches on bead geometry estimation. Such chapter had its papers searched, selected and studied through a systematic approach (FERENHOF; FERNANDES, 2018).

The state of the art research was set to be a combination of two different research queries, the first focused on optical monitoring systems for the laser cladding process, and the later, on artificial intelligence applied to the process. Both research queries were applied on the Scopus, Web of Science and Compendex databases. Those queries can be seen below in Figure 6.

Figure 6 - Search queries for optical system monitoring (left) and artificial intelligence for laser cladding (right).

|   |  |
|---|--|
| ( "laser cladding" OR "laser deposit" OR "laser deposition" OR DMD OR LBAM OR LMD OR "metal deposition" ) | ( "laser cladding" OR "laser deposit*" OR DMD OR LBAM OR LMD OR "metal deposition" ) |
| AND   | AND  |
| ( camera* OR ccd OR cmos OR video )   | (geometr* OR profil* OR shap* OR heigth)   |
| AND   | AND  |
| (pool OR clad OR bead* OR track* OR melt* OR molte*)  | (measur* OR monitor* OR metrology OR sensor* OR control* OR optimiz*)                |
| AND   | AND  |
| (geometr* OR profil* OR shap* OR heigth)  | ("neural network" OR AI OR "machine learn*")   |
| AND   |  |
| (measur* OR monitor* OR metrology OR sensor* OR control* OR optimiz*)                                     |  |

On the first research, by the date 27/02/2018, the Scopus database returned 74 entries, Web of Science returned 49 and Compendex, 71, with

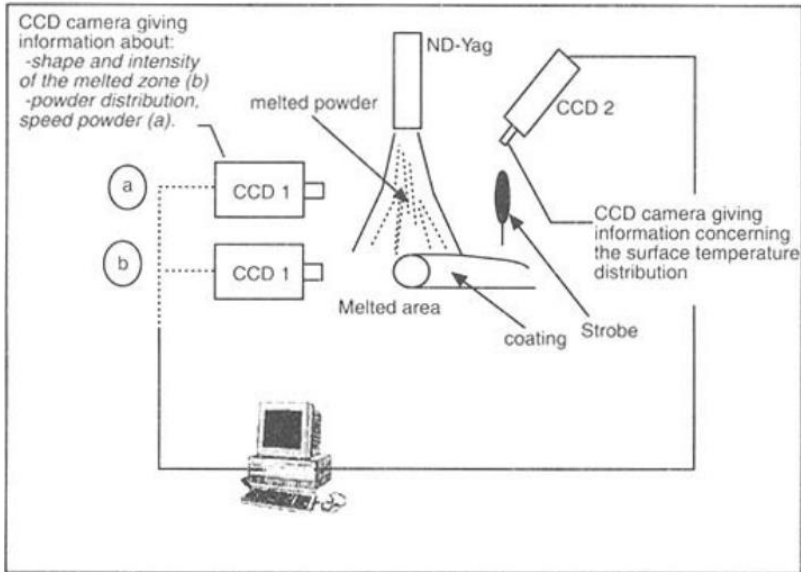
a total of 194 results, 91 eliminating duplicates. After filtering by interest field, 24 entries remained, but only 17 of those had full text available.

On the second research, by the date 27/11/2018, the Scopus database returned 27 entries, Web of Science returned 11 and Compendex, 35, with a total of 73 results, 48 eliminating duplicates. After filtering by interest field, 9 entries remained, but only 6 of those had full text available. One more paper was incorporated to complement the results, based on exploratory research. Both research queries had one entry in common.

This chapter presents works on those fields dating from 1996 to 2018, in chronological order.

Meriaudeau et al (MERIAUDEAU; RENIER; TRUCHETET, 1996; MERIAUDEAU, FABRICE; TRUCHETET, FREDERIC, 1996; MERIAUDEAU, F.; TRUCHETET, F., 1996; MERIAUDEAU *et al.*, 1996), in his four papers from the year 1996, used two CCD cameras directed to the molten pool, one in a shallow, almost horizontal angle, and the other on a 45° angle, to acquire process information.

Figure 7 - System with two cameras, one on a shallow angle and the other on a 45° angle

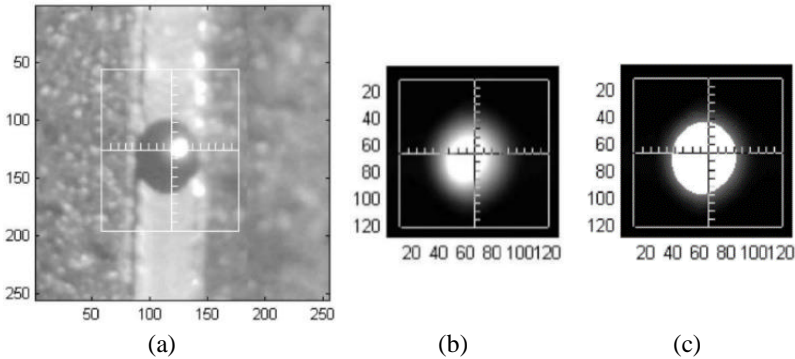


Source: (MERIAUDEAU, F.; TRUCHETET, F., 1996)

This camera positioning makes this system dependent on cladding direction. The shallow angle camera acquired the bead geometry during its formation. Bead height and width were measured in real time with a simple edge detection algorithm. The powder flow was also observed from this camera. The 45° camera was used as a spectral thermometer, converting the image luminosity (gray level) on temperature, as there is a linear relationship between the two quantities. On the resulting image histogram, two peaks can be seen, one relating to the image background color, and the other to the molten pool temperature. System resolution was of 5°C, and maximum errors of 15°C. Future works aimed at powder particle speed acquisition.

Hu et al (HU; MEI; KOVACEVIC, 2002; HU; KOVACEVIC, 2003) used two CCD cameras, one capturing infrared (IR) images coaxial to the laser beam, and the other being a high-speed camera (800 fps), on a near vertical angle. This camera is synchronized with a short-wavelength stroboscopic illumination, yielding a clear bead geometry image, on Figure 8a. The IR camera is filtered with a long pass 700 nm optical filter, acquiring only infrared frequencies, related to temperature. The resulting image gray level can be related to temperature, but not a precise contour for the molten pool can be seen (Figure 8b). Merging both images reveals the molten pool borders on the IR image. A grey level threshold can then be defined, relating to the material's melt point isotherm, as in Figure 8c. The molten pool is then segmented. The number of pixels inside the molten pool is counted and used as the control variable in a closed loop. A PI controller successfully controls the molten pool size by varying the laser power.

Figure 8 - Molten pool images: (a) high-speed camera image; (b) infrared coaxial camera image; (c) molten pool isotherm



Source: (HU; KOVACEVIC, 2003).

Toyserkani et al (TOYSERKANI; KHAJEPOUR, 2006) utilized a CCD camera on a shallow angle and a halogenic lamp for a side image of the molten pool. Once again, this disposition choice makes this system dependent on cladding direction. This camera was equipped with a magnifying lens and many optical filters. After the acquisition, the image has its brightness and contrast adjusted and is converted to gray level. It is then binarized with a fuzzy threshold technique, also developed by the authors. Finally, the molten pool is segmented. This side view of the molten pool allows the measurement not only of the bead height but the solidification angle as well, making it possible to estimate the generated microstructure. System resolution is of 0.02 mm for bead height, with a 0.1 mm uncertainty. A PID controller closes a loop for bead height control by varying laser power. This control loop enhanced bead quality by eliminating disturbances during the process. Future works aimed at adapting the system to work with any cladding direction, incorporating a total of 3 CCD cameras on a radially symmetric arrangement, centered at the molten pool.

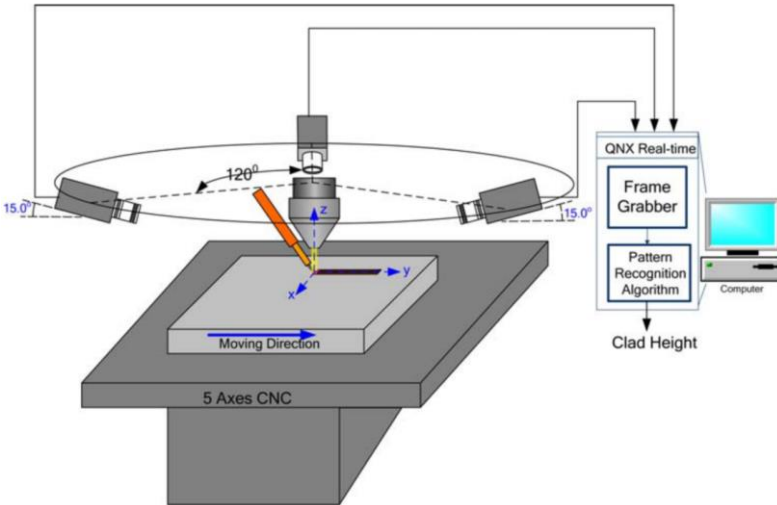
Xing et al (XING; LIU; WANG, 2006) measured molten pool temperature through colorimetric methods. The system has an alternating filter device with two bandpass filters (790 nm and 921 nm). Those filters are alternated in front of the CCD camera synchronously to the camera frames. The ratio between the different image intensities for each frequency allows temperature measurement. Moreover, a laser line projection allows bead height measurement by the light triangulation method. Once the molten pool temperature is known, it is successfully

controlled by a fuzzy logic controller, acting on both laser power and travel speed. Future works aimed at implementing stochastic models and neural networks.

Hofman et al (HOFMAN; DE LANGE; MEIJER, 2006) developed a camera-based feedback control system for the laser cladding process, aiming at energy usage optimization. The system consists of a coaxial CMOS camera arrangement, which is optically filtered to gather mostly infrared (temperature related) wavelengths. Then, the image is processed with an algorithm that can be described as blurring, thresholding for molten pool boundary identification, ellipse fitting, ellipse feature extraction – area, length, width, and rotation angle –, and the pixel to mm and mm<sup>2</sup> conversion. It was observed that there is an almost linear relationship between the molten pool width and the bead width, although there is no apparent relation between bead height and molten pool width. It was also observed that there is a molten pool width threshold. While the molten pool is narrower than this threshold, molten pool depth is nearly zero. After that limit, it increases rapidly. For the control logic, the molten pool width is used as the control variable while laser power is varied to compensate heat effects, as an effort to keep molten pool width constant. A digital PID controller is thus implemented, successfully achieving minimal dilution.

Iravani-Tabrizipour et al (IRAVANI-TABRIZIPOUR; ASSELIN; TOYSERKANI, 2006; IRAVANI-TABRIZIPOUR; TOYSERKANI, 2007) continued Toyserkani's research (TOYSERKANI; KHAJEPOUR, 2006) developing a system with 3 CCD cameras on a radially symmetric arrangement, centered at the molten pool, as seen in Figure 9, achieving the proposed clad direction independence. Magnifying lenses and bandpass 700±40 nm filters were equipped on the cameras. On each moment, only two out of the three camera images are processed, depending on actual cladding direction, to reduce total image processing. The molten pool is segmented from the images with the same fuzzy threshold method from the previous work. Each molten pool image is fitted to an ellipse, then, merged with a coordinate transformation. The ellipse features are fed into a recurrent neural network (Elman RNN) that outputs bead height. This network was trained by the error backpropagation algorithm. Its structure naturally diminishes noise influence. The average error is 12.5%, and the system uncertainty is around 0.15 mm. Future works aim at bead height measurements for curved tracks.

Figure 9 - Schematic of the trinocular CCD based detection system.



Source: (IRAVANI-TABRIZIPOUR; TOYSERKANI, 2007).

Lei et al (LEI; WANG; LIU, 2010) achieved a molten pool image acquisition on a CO<sub>2</sub> laser-based cladding process. The biggest challenge for this type of laser is the non-transparency of glasses for its wavelength, thus, a coaxial camera coupling is not possible. The camera was then coupled on a 25° angle from the laser beam. The acquired molten pool was fitted to an ellipse, and through its brightness, it was possible to measure molten pool temperature.

Mondal et al (MONDAL; BANDYOPADHYAY; PAL, 2010) aimed at finding a relation between bead geometry (height and width) and process input parameters (laser power, travel speed, and powder flow). To this end, many beads were cladded with different process parameter values. Those values were chosen according to the Taguchi design of experiment method. The beads were cut, and their cross section, measured. With the values of height, width, laser power, travel speed and powder flow for each bead, this data was fed into an artificial neural network (ANN), where the process parameters were the input, and the bead geometry, the output. A deviation occurs between prediction and target values due to modeling or experimental errors, yielding an R<sup>2</sup> of 0.981 for the best fit line. This result shows that the predicted values were in great agreement with the experimental outcomes.



Barua et al (BARUA; SPARKS; LIOU, 2011) presented a low-cost alternative for the molten pool visualization problem. According to the black body theory, there is no ultraviolet radiation emitted from the laser cladding process. Ultraviolet LEDs were then used to illuminate the molten pool. A CMOS webcam, optically filtered with a UV bandpass filter, acquired the images. The whole setup can be seen in Figure 10. The CMOS technology was found to be more sensitive to the UV spectra than the CCD technology. After preprocessing, segmentation and perspective correction, the molten pool was measured. The above-mentioned algorithm was developed with the OpenCV library. Information extracted from the image includes molten pool size and circularity. Measurement errors were identified coming from an automatic white balancing function of the webcam. Also, a loss on the resolution was identified, what could be corrected by stronger illumination. Future works aim at a control system with the Labview language to reject process disturbances, as well as pore detection and bead height monitoring.

Figure 10- Experimental setup with a webcam and UV illumination.



Source: (BARUA; SPARKS; LIOU, 2011)

Davis et al (DAVIS; SHIN, 2011) utilized a CCD camera and a laser plane to measure the height of the newly formed bead by the laser triangulation principle, 5 mm after the molten pool. The laser plane and the camera are angled at  $55^\circ$ . To avoid measurement errors, it was observed that the perpendicularity between the bead and the laser plane is crucial. Even at higher speed, the system errors are about only  $50 \mu\text{m}$ . Future works consist of a closed loop control system with the LabView language.

Liu et al (LIU; WU; WANG, 2012) approached the molten pool segmentation problem by a previous calibration methodology. This methodology consists of previous laser cladding of beads. Those beads are then cut, and their cross sections measured, acquiring bead width. Theoretically, bead width equals molten pool width. This width value allows a gray level threshold value to be set on the image, segmenting the molten pool. The image acquisition system consists of a CCD camera on a 20° angle from the laser beam. Errors on this methodology vary from 0.56% to 5.33%, which came from image scaling, thresholding or vibrations.

Doubenskaia et al (DOUBENSKAIA *et al.*, 2013) present a methodology for molten pool temperature distribution acquisition by black body calibration. The system consists of an infrared camera on a 40° angle from the laser beam. In this work, the image emissivity is calculated from the solid-liquid interface of the molten pool, where the melting point is known. The system successfully acquired process temperature distributions and cooling rates.

Arias et al (ARIAS *et al.*, 2014) developed a system with a dynamic laser spot size. Coaxially to this system, there is a CMOS camera with a final resolution of 10  $\mu\text{m}$ . The optical filters on the system allow only the wavelengths between 2450 and 950 nm to be acquired. This system is FPGA-based, making use of all its speed and processing capacity. The image processing algorithm is based on blob detection, where the biggest blob on the image refers to the molten pool. Features extracted from the blob include blob center and size. With the biggest blob width measurement corresponding to the molten pool width, it is possible to implement a control loop when altering laser power.

Ocylok et al (OCYLOK *et al.*, 2014) utilized a coaxial CMOS camera to acquire molten pool images. A threshold operation was enough to segment the molten pool. Molten pool width, length, and area were measured. The author presents a study on the influence of process parameters (laser power, travel speed, and powder feed rate) on molten pool geometry. Conclusions are that the least affected molten pool dimension is its area, apart from laser power, with which this dimension is linearly related.

Moralejo et al (MORALEJO *et al.*, 2017) developed a closed loop control system to control molten pool geometry in real time. For this purpose, the authors developed a PI controller with feedforward action based on the molten pool width measurement from a coaxial CMOS camera. A derivative control action was not considered due to powder

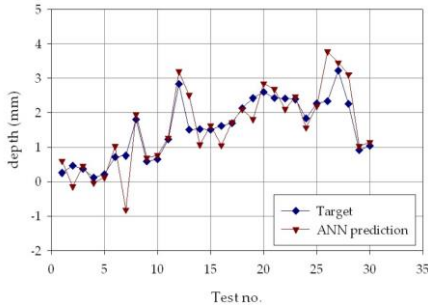
noise. The feedforward action became necessary to speed up the PI response. Molten pool width was the most stable molten pool dimension; therefore, it was chosen as the control variable. An investigation on optical filters leads to an optimal combination consisting of a notch filter on the laser beam frequency (1064 nm) and a long pass filter of 700 nm. The threshold value was chosen from images of the process without powder flow. For validation, a part with width varying on a sinusoid shape was clad. The system presented a standard deviation of 3.5 pixels.

Aggarwal et al (AGGARWAL; URBANIC; SAQIB, 2018), targeting at bead geometry optimization, developed predictive models to select input process parameters on both single and overlapping laser clad beads. On this work, three approaches were taken – one experimental, the second on predictive models, and lastly an artificial neural network (ANN) approach with the MATLAB toolkit. The experimental approach consists of varying a set of five input parameters over 5 levels, on the single bead case and on the 40, 50 and 60 percent overlap cases. The data here acquired was also used on the following approaches. The predictive model approach was subdivided into another two methods: an ANOVA approach (quadratic model) and a physics-based model related to travel speed, laser power, as well as observed data trends. Finally, an ANN approach is taken, inputting the desired geometry and resulting in the appropriate process parameters. Results show that the classic method for this problem, the ANOVA analysis, yielded the worst results, while the ANN results have 96.3% of confidence level, the best of the approaches. Future works direct to the study of different material, dilution minimization, and bead width variation while keeping constant bead height, for complex shapes.

Caiazza et al (CAIAZZO; CAGGIANO, 2018) developed an ANN-based process parameters estimation method. Acquiring cross-section data, each set of bead height, width, and depth was joined with its corresponding process parameters (laser power, travel speed, and powder flow rate) and fed into a three-layer cascade-forward backpropagation ANN. Different neural network architectures were tested aiming at finding the optimum network for the system. The Levenberg-Marquardt algorithm was chosen as the ANN training function. A total of 90 samples were used for training, and the evaluation occurred in terms of root mean square error between predicted and target values. On the first phase, the neural network was used to predict bead geometry from the process parameters. Once the architecture was selected, inputs and output were

reversed, estimating process parameters from bead geometry. Results show that the ANN was able to accurately estimate the correct process parameters necessary for the desired bead geometry, with mean absolute percentage errors as low as 2% for laser power, 5.8% for travel speed and 5.5% for powder feed rate. An example of this ANN estimation can be seen in Figure 11.

Figure 11 - Experimental average and ANN predictions for depth.

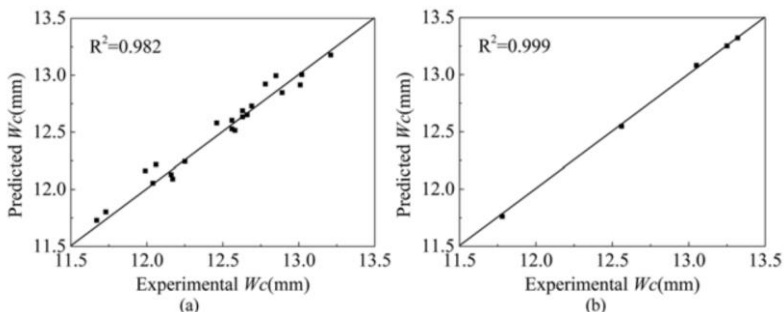


Source: (CAIAZZO; CAGGIANO, 2018)

Huaming et al (HUAMING, 2018) predicted the geometric characteristics from the input process parameters with a genetic algorithm and backpropagation neural network-based approach (GA-BPNN). For the data, experiments were performed varying sets of laser power, travel speed and powder thickness over three levels, on a pre-placed powder setup. After the cross-section acquisition, three main parameters were chosen as outputs, being bead height, width and contact angle. After acquiring the data, a genetic algorithm is used to optimize a backpropagation neural network architecture. Each population consists of different neural networks, which are trained and tested. The fittest network is chosen to generate a new population of networks, a process that continues for 50 generations. Before optimization, the neural network took 40 epochs to achieve its best training performance. After optimization, it only took 12 epochs for much better performance. The best results were on width predictions, where the  $R^2$  factor of the scatter graph between prediction and target values was 0.982 on training and 0.999 on testing, as seen in Figure 12. Conclusions point to the GA-BPNN approach being an effective tool to correlate process parameters and bead geometry, with the ANN error being significantly reduced after GA optimization. It was also observed that the double hidden layer architecture has a smaller relative error than the single hidden layer one.

Additionally, a network outputting only a single parameter performs much better than one with multiple outputs.

Figure 12 - Scatter diagram with the best fit of GA-BPNN prediction vs. experimental. (a) training patterns (b) test patterns.



Source: (HUAMING, 2018)

It can be observed a difference in the papers before and after the year 2010. The early works rely heavily on hardware, often using more than a single camera or demanding external lighting. The software development was limited and with low complexity. Despite those limitations, the information there acquired was crucial for this technology's development in the later years. From the year 2010 onwards, different approaches were taken with significantly less hardware. The post-processing analysis of cross-sections became a trend. However, most of the analysis relied solely on cross-section measurement, which implies a single measurement for the whole clad. Information on process oscillations is, thus, lost.

Those approaches have reported both image acquisition followed by processing, or cross-section measurements fed to machine learning algorithms. An approach where both data are fed into a machine learning algorithm, especially to state-of-the-art structures such as convolutional neural networks, was not present in this research. Such an architecture is capable of taking the raw image as an input, without the need for extensive preprocessing, e.g. to fit the molten pool to ellipses, to acquire any molten pool length or width nor to measure any areas. The very architecture is responsible for defining the main features to be identified on the image for further geometry values estimation. Besides that, such an approach requires nothing more than a computer, a camera, and filters. The implementation of this approach is explained in the following chapter.



## 4 Implementation

This chapter details how each step on this work is performed. For neural networks to be developed and trained, many steps needed to be taken. It starts with an analysis of the initial optical setup and which components needed to be replaced for image acquisition, as well as setup adequation, which are detailed in section 4.1. Next, the laser clad bead deposition process is described in section 4.2. Later, it is discussed the procedure on clad beads geometry measurement, in section 4.3. After that, the image preprocessing is described in section 4.4. Finally, all necessary data is now available, providing the networks with both inputs (molten pool images and process parameters) and outputs (geometry measurements). the choices on language and libraries are presented for CNN development, as well as metrics are defined for its performance evaluation, and its different architectures, compared, in section 4.5.

### 4.1 Optical system adequation

Prior to this work, the system contained a coaxial powder cladding head, with such an optical setup as described and depicted on section 2.1.1, along with a manual aperture close to the beam splitter on the camera's optical path. This camera, however, was an analog camera, model CF 8/5 MX from the Kappa company, on Figure 13a. With a series of embedded functions to improve image visualization on different light intensities, this camera is a perfect match for its current purpose, which is to watch the process molten pool on a monitor screen. However, acquiring digital images from it has proven not to be the easiest task.

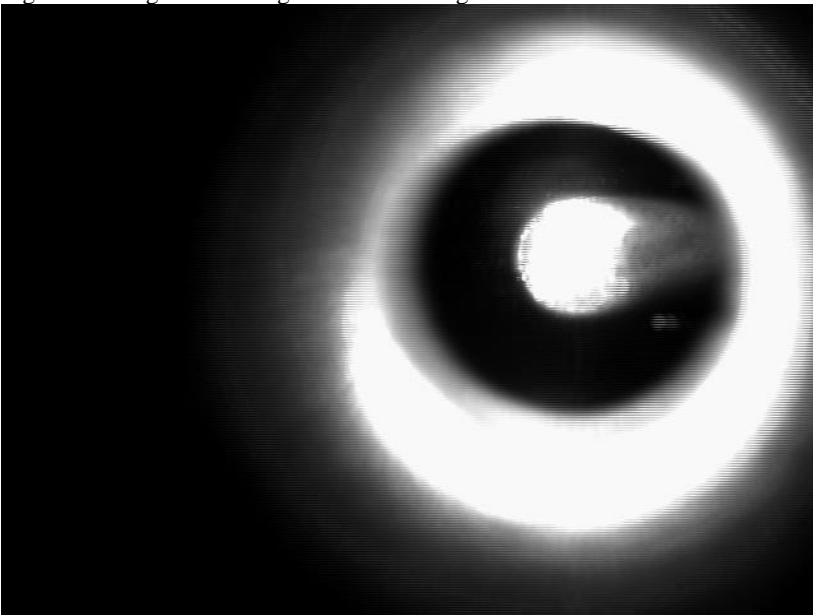
An approach has been made to digitalize this analog camera's image. For this purpose, the Video-to-USB 2.0 converter on Figure 13b, from The Image Source company, was used. To do so, all image improving functionalities from the camera were shut off, keeping the image constant even when there was variable light intensity. The converter digitalized the image to a size of 640x480 pixels, on a frequency of 30 fps. A digitalized image sample from this setup can be seen in Figure 14.

Figure 13 - Analog camera, model CF 8/5 MX from Kappa (a). Image converter, model Video-to-USB 2.0 converter from The ImageSource (b).



Source: (IMAGINGSOURCE; KAPPA, 2006)

Figure 14 - Digitalized image from the analog camera.



Source: Author.

This is the image from the inside of the laser cladding head. The molten pool can be seen in the middle, while the white halo around it represents laser and other sorts of reflections on the inside of the cladding head. The camera is off-center at this image, but the whole molten pool is



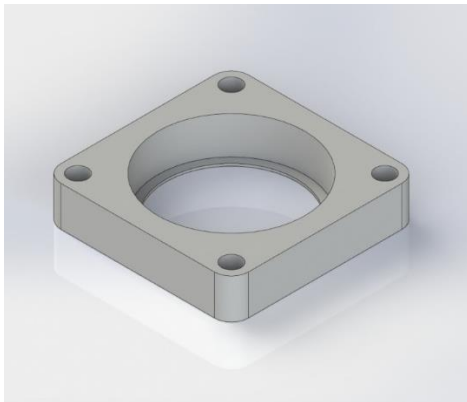
still visible. The high saturation levels on the image can also be noted, as well as how small the molten pool is when compared to the whole image.

From this image, the interlacing effect can be observed. It happens because of the digitalizing process, where even-numbered and odd-numbered pixel rows are alternated to form consecutive frames, resulting in horizontal lines. It can also be seen that there is an image duplication, as if two images were horizontally misaligned, blurring the near vertical edges. This does not happen due to the analog camera but to system innate characteristics.

After acquiring those images, some improvements were planned:

- Reduce saturation with optical filters: A filter support was manufactured by FDM, illustrated in Figure 15. It was placed below the camera, without the need for further adaptations;
- Increase image magnification: An optical system to magnify the image was designed and then manufactured through FDM, but it was concluded that due to machine vibrations, focusing difficulties, and lack of system rigidity make it unpractical and unusable;
- Replace the analog camera with a digital camera – A digital camera, model BFLY-PGE-20E4M-CS from PointGrey, with a resolution of 1600x1200 and 50 fps, replaced the analog camera.

Figure 15 – Designed filter support. It fits up to two 25.4 mm (1 inch) diameter filters.



Despite the magnification, all improvements were implemented. The increased resolution of the digital camera was used to compensate for the magnification absence. This new system is now able to acquire images at much higher speed and resolution than the previous one, besides having a steadier light level than when using the analog camera. There remained a certain saturation level, although it kept away from the borders of the molten pool, which was considered acceptable.

#### 4.2 Laser clad processing

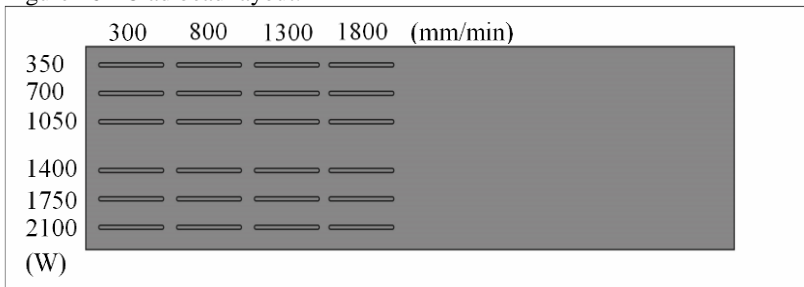
After setup adequation, the clad beads could be deposited. Feeding powder consisted of AHC 100.29 Höganäs manufactured iron (99,98%) powder, fed with 6.51 g/min. The substrate is composed of ASTM A36 steel, with the 50 mm x 200 mm x 9.52 mm dimensions. Clad beads were 20 mm long. Laser power and travel speed values can be seen below in Table 1. Those values were chosen based on previous experience taken from other laboratory research lines.

Table 1 - Input process parameters. Source: Author

| Laser Powers (W)       | 350 | 700 | 1050 | 1400 | 1750 | 2100 |
|------------------------|-----|-----|------|------|------|------|
| Travel Speeds (mm/min) | 300 | 800 | 1300 | 1800 |      |      |

Beads were placed over the substrate according to Figure 16. On this figure, each column has a different speed value, while each row represents a different laser power value. The whole experiment was repeated 3 times to increase data volume, resulting in a total of 72 clad beads.

Figure 16 - Clad bead layout.



A top view of the cladded beads is depicted in Figure 17, in which the top left 24 clad beads (in blue) do not make part of this work, as they

were performed without powder. However, the remaining three sets of 24 clad beads are, therefore, this work's object of study, each of them following the parameter distribution presented previously in Figure 16 and highlighted in red in Figure 17.

Figure 17 - Clad beads. Clads deposited on red rectangles correspond to the three sets of data used in this work. Marks on the blue rectangle were made without powder.



It can be observed some misaligned clads on the top rows, corresponding to 350 W for laser power. On those beads, there was not enough energy to weld them on the substrate, thus leading to their detachment. Later on, those beads were glued back to their positions to allow further analysis. Such an unusual procedure did introduce errors in measurement, as will be explained in the next section.

The images were acquired with the Spinview software, also from PointGrey, with its gain set to 15. The aperture was adjusted, though no numerical value could or can be read to quantify it. Metallic neutral density filters from Newport were used, with combined optical densities of 2. The Maximum Transmit Unit (MTU) of the connection between camera and computer was set to 9000.

Once the process is done, the next step clad beads measurement, on section 4.3

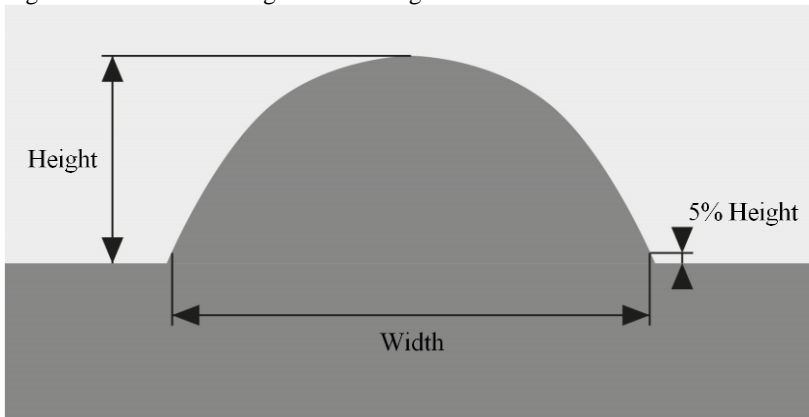
### 4.3 Clad bead geometry measurement

The clad bead consists of a single 20 mm-long cladded line. Although simple in geometry, the clad bead characteristics vary greatly from one another. Each clad has a height and a width. On this work, those dimensions are defined as follows:

- Height – the biggest distance between substrate level and the clad bead surface, perpendicular to the substrate
- Width – the biggest distance between points on the clad bead surface that have at least 5% of the clad bead height, parallel to the substrate.

Those dimensions are depicted in Figure 18.

Figure 18 - Clad bead height and width geometries.



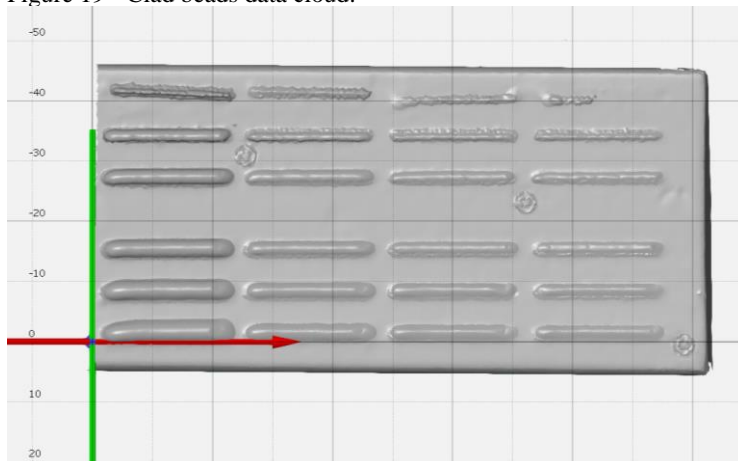
After the beads are manufactured, they are measured to extract its height and width dimensions. For such a task, an ATOS Gom system (Compact Scan model) was used. This system scans 3D parts based on active photogrammetry with fringe projection. The result is a dense data cloud in the shape of the part's surface, seen in Figure 19.

For acquiring clad bead height and width values, the first step is to position the coordinate system. It was set with the X-axis parallel to the clad beads, not to the side of the substrate, as both are not perfectly aligned. The Y-axis is then placed so that the XY-plane is coincident to the surface of the substrate.

It can also be observed that most beads cladded with the lowest power (top row of Figure 19) are not parallel with the remaining beads, even missing parts when cladded with the highest speed (top right bead on

Figure 19). Those were beads without any metallurgical bond, thus, they have detached. This displacement does not interfere much on the height dimension but does introduce an error on width. Without any metallurgical bond, those are not desirable geometries, thus, those errors can be neglected. This is also the case for the missing part of the top right clad bead. However, they do introduce errors on the following neural networks training. An attempt to completely dispose of those beads was made, although leading to worse results.

Figure 19 - Clad beads data cloud.



The clad beads need to be sectioned. During the process, the camera acquired a number of frames for each bead. Each of those frames needs a matching height and width value. Each bead was then sectioned into the theoretical number of frames for its speed, which is calculated by dividing its length by its travel speed and then dividing this result by 0.02 seconds (50 fps). The theoretical number of frames for each speed is presented in Table 2, while the beads with their sections are shown in Figure 20.

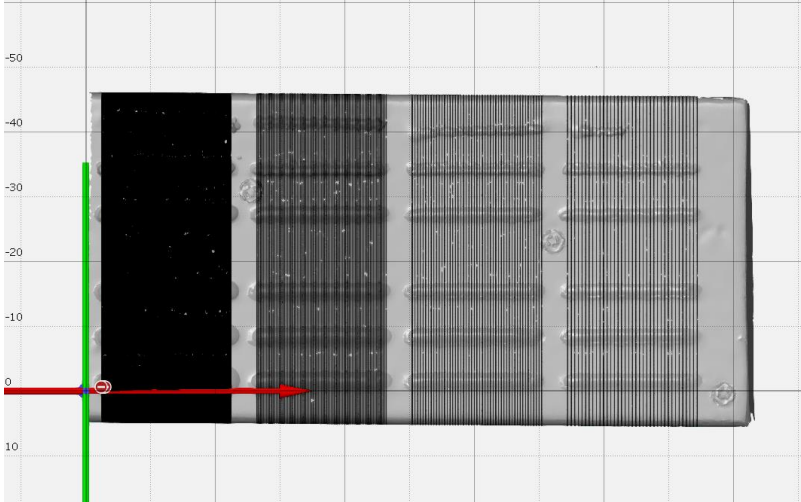
Table 2 - Theoretical number of frames for each travel speed.

| Travel Speed (mm/min)        | 1800 | 1300 | 800 | 300 |
|------------------------------|------|------|-----|-----|
| Theoretical Number of Frames | 18   | 48   | 78  | 108 |

The extracted data consists of 3-dimensional coordinates of every point belonging to each section. Then, they were split into sections of individual clad beads, and their height and width dimensions, calculated

according to the definition in Figure 18. Finally, the numbers of frames and sections for each bead are manually evened. This matching is not precise, introducing errors on the following network's training as well.

Figure 20 - Clad beads with sectioning.



The clad beads are now measured, leaving only the images to be preprocessed before all data can be fed to neural networks. The image preprocessing is discussed in the following section, 4.4.

#### 4.4 Image preprocessing

During the process, there was no synchronized mechanism to turn the camera on with the laser. So, the camera kept recording during the whole process, acquiring images from all clad beads. This video file needed then to be split into individual clad beads. After that, it was again split into training and testing sets. As the set of 24 different clad beads was repeated 3 times – resulting on 72 clad beads – one of these sets was kept as testing data.

As the camera kept recording in between clad beads – when the laser was off – there were many black frames. Those were filtered out based on image pixel mean value. Every image with its mean pixel value below 1 was discarded. This filtering discarded 1584 images out of a total of 5956 images from the training data, remaining 4372 images. For the test data, 484 images were deleted, with 2189 images remaining. A sample of the image after this selection can be seen in Figure 21.

As can be observed, most of the image is a black background. This area was simply cropped out of the images, resulting in the cropped image area in Figure 21, of size 540 x 540 pixels. Then, a small area between the nozzle area and the cropped image borders remaining. This area was masked out, as the pixels there represent reflections on the inside of the nozzle, which can be considered noise. Finally, the image was scaled down to 128 x 128 pixels to save processing time and space.

All of this image preprocessing was performed on the Python 3.6 language through the ipython platform. The libraries used for such are OpenCV and Numpy, both open source and widely available. The Glob library was also used for file management. The final result is in Figure 22. On this stage, the image is ready to be processed by convolutional neural networks. No further image preprocessing is needed. The next step is to develop those networks, on section 4.5.

Figure 21 - Image from training set before preprocessing. Nozzle area and cropped image regions. 1050W, 300mm/min. 1200 x 1600 pixels.

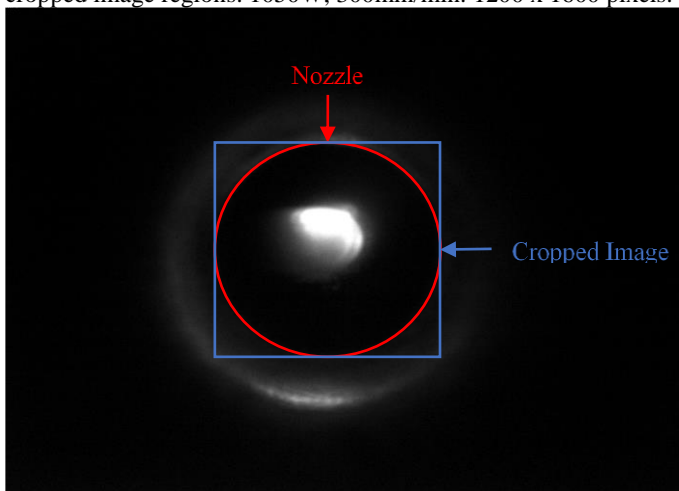
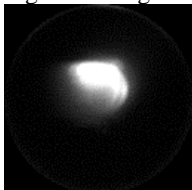


Figure 22 -Figure 21 after cropping, masking and rescaling. 128 x 128 pixels.



## 4.5 Neural network development and evaluation

For developing convolutional neural networks (CNN), the Python language was chosen due to its many available libraries on the machine learning field. One of those many libraries is Keras – here used with the Tensorflow backend – presenting a simple interface for developing the CNN. However, before the network could be developed, the data needed to be adjusted for such. Image, input parameter, and output of each frame need to be related to each other. This is performed by a custom image generator, explained in section 4.5.1. With the input defined, the main body of the convolutional neural network can be developed, on section 4.5.2. Finally, the methods used for convolutional neural network evaluation are described in section 4.5.3.

### 4.5.1 Image Generator

To set images as input for a convolutional neural network (CNN), the Pandas library was used. Two CSV files relating the image file paths, input process parameters and output geometry values were used to create pandas dataframes, one for training and the other for testing, on lines 1 and 2 of **Erro! Autoreferência de indicador não válida..** On line 4, a standard image generator is created, which will load the images and rescale their pixel value from the 0-255 range to the 0-1 range. In line 6, the proper custom image generator is created. Lines 36 and 37 show its usage, with the standard generator, dataframe, path and shuffle mode as arguments.

Lines 7 to 18 are the call to the `flow_from_dataframe` method of the Keras `ImageDataGenerator` class (VIJAYABHASKAR, 2018). This method purpose is to select the images contained on a provided dataframe – along with its related numerical values, here input parameters and output values – and feed it in batches to the CNN. The arguments for this method are explained in the library documentation (KERAS, 2018).

The highlights are the `x_col` and `y_col` arguments, which represents respectively the inputs and the outputs of the neural network, although the method does not support more than one input type. Those arguments are set, therefore, with the image file as the input (`x_col`) and both process parameters and output values as outputs (`y_col`). This will be corrected later on the code.

Another point to highlight is the `class_mode` argument, which is related to the neural network operation. There is no specific value to use



on the regression operation, so “other” is recommended by the software development community. This is what sets the neural network to operate a regression, not a classification operation.

Figure 23 - Image generator code.

```

1. train_df=pd.read_csv(top_path+"train.csv", sep=';')
2. test_df=pd.read_csv(top_path+"test.csv", sep=';')
3.
4. imgen = ImageDataGenerator(rescale = 1./255)
5.
6. def generate_generator_multiple(generator,df, path, shuffle):
7.     genX1 = generator.flow_from_dataframe(
8.         dataframe=df,
9.         directory=path,
10.        x_col=filename,
11.        y_col=outputs + inputs,
12.        has_ext=True,
13.        batch_size=batch_size,
14.        shuffle=shuffle,
15.        class_mode="other",
16.        color_mode='grayscale',
17.        target_size=IMAGE_SIZE
18.    )
19.    while True:
20.        X1i = genX1.next()    # img, [inputs, outputs]
21.        i1 = np.array(X1i[0]) - 0.5
22.        labels = np.array(X1i[1])
23.        o = labels[:,0:len(outputs)]
24.        i2 = labels[:,len(outputs):len(outputs + inputs)]
25.
26.        # normalization
27.        # (x - xmin) / (xmax - xmin) - 0.5
28.        normP = np.vstack(((i2[:,0])-
29.        np.full(i2[:,0].shape, min_inputs[0])/
30.        (max_inputs[0]-min_inputs[0])-0.5)
31.        normS = np.vstack(((i2[:,1])-
32.        np.full(i2[:,1].shape, min_inputs[1])/
33.        (max_inputs[1]-min_inputs[1])-0.5)
34.        i2 = np.hstack((normP, normS))
35.        yield [i1, i2], o    # [img, inputs], outputs
36.
37. train_generator=generate_generator_multiple(generator=imgen, df=train
in_df, path=train_path, shuffle=True)
38. test_generator=generate_generator_multiple(generator=imgen, df=test
_df, path=test_path, shuffle=False)

```

Finally, the shuffle mode argument differs between the training and the testing phases. It is desirable for the training data to be as random as possible, although the testing data should be kept in order to evaluate the network always on the same way, this explains the shuffle values on lines 36 and 37.

The data generator yields data in the shape of (inputs, outputs). Due to the x\_col and y\_col values set earlier, the result has the shape of

(inputs=image, outputs=[parameters, dimensions]). If that was correct, the process parameters would be set as network outputs, which is not their intended role. The correct shape is (inputs=[image, parameters], outputs=dimensions). That way, the neural network can be fed with the inputs and learn from the outputs correctly. The lines 19 to 34 correct such.

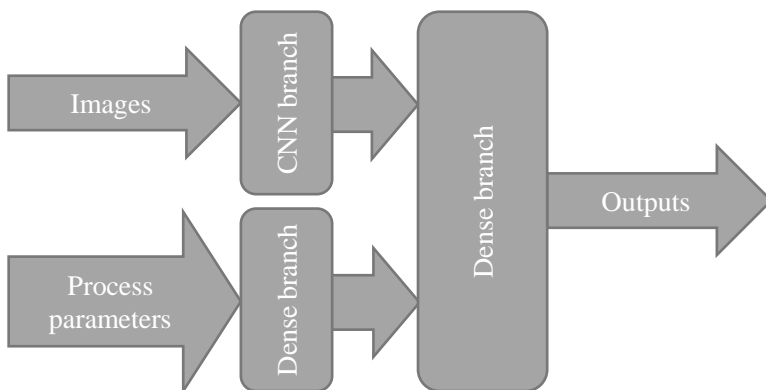
The while loop on line 19 combined with the yield statement on line 34 transform this method on yet another generator. It starts on line 20 by calling the generator created on line 7. On line 21, the images pixel values are reduced by 0.5, which results on they ranging between -0.5 and 0.5, thus being normalized. Line 23 extracts the dimensions (outputs) values, which does not need any more preprocessing. Line 24 extracts the input parameters, which need normalization, done in lines 28 to 33 according to the formula on line 27. After that, all inputs – images and process parameters – have their values in between the -0.5 to 0.5 range. Finally, the result is yielded on the shape of (inputs=[image, parameters], outputs=dimensions), a tuple with an array of the actual inputs as the first element, and the array of outputs as the second element.

With the inputs and outputs correctly arranged, the main body of the convolutional neural network can be developed, on section 4.5.2.

#### **4.5.2 Convolutional Neural Network Layers**

The convolutional neural network (CNN) architecture consists of two branches, one for the image input and the other for process parameter inputs. A schematic can be seen in Figure 24.

Figure 24 - Schematic for CNN architecture



In this schematic, it can be observed that the network has two different input types, both images and process parameters. The code for the convolutional branch – the one that processes the images – can be found in Figure 25, while the dense one for the parameter branch, in Figure 26. Both branches are then merged into one dense layer, which its code is presented in Figure 27. The outputs of the network appear from

the two output neurons: one for the width dimension, the other for the height one.

Figure 25 – Image branch code.

```

1.  actfunc = 'linear'
2.
3.  i1 = Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 1))
4.
5.  x = Conv2D(filters=8, kernel_size=(3, 3), padding='same')(i1)
6.  x = BatchNormalization()(x)
7.  x = Activation(actfunc)(x)
8.  x = MaxPooling2D()(x)
9.
10. x = Conv2D(filters=16, kernel_size=(3, 3), padding='same')(x)
11. x = BatchNormalization()(x)
12. x = Activation(actfunc)(x)
13. x = MaxPooling2D()(x)
14.
15. x = Conv2D(filters=32, kernel_size=(3, 3), padding='same')(x)
16. x = BatchNormalization()(x)
17. x = Activation(actfunc)(x)
18. x = MaxPooling2D()(x)
19.
20. x = Conv2D(filters=64, kernel_size=(3, 3), padding='same')(x)
21. x = BatchNormalization()(x)
22. x = Activation(actfunc)(x)
23. x = MaxPooling2D()(x)
24.
25. x = Conv2D(filters=128, kernel_size=(3, 3), padding='same')(x)
26. x = BatchNormalization()(x)
27. x = Activation(actfunc)(x)
28. x = MaxPooling2D()(x)
29.
30. cnn = Flatten()(x)

```

The image branch has an input layer. It is then followed by sets of convolutional, batch normalization, activation, and pooling layers – convpool layers for short – ending with a flattening layer. The code for this CNN branch can be seen in Figure 25. Every convolutional layer has the same kernel size and the same padding type – which were defined experimentally – however, different quantities of filters. All neural network architectures here experimented have the first three convpool layers on lines 5 to 18. Architectures with 4 convpool layers also have the layers on lines 20 to 23, and the ones with 5 convpool layers include all the previous ones.

The parameter branch, however, has a single set of dense and activation layers after its input layer. The number of neurons here was chosen as the same number of neurons from the imaging branch after flattening, aiming to balance both branches. The code for this branch can be seen in Figure 26.

Figure 26 - Input parameters branch code.

```

1. i2 = Input(shape=(len(inputs),))
2. x = Dense(units = 2048)(i2)
3. dense = Activation(actfunc)(x)

```

Both branches are concatenated together, then followed by sets of dense, activation and dropout layers, with a fixed number of neurons. The last pair of dense and activation layers returns the result of the network, in Figure 27.

Figure 27 - Merging branch code.

```

1. x = concatenate([cnn, dense])
2. # x = Dense(units=500)(x)
3. # x = Activation(actfunc)(x)
4. # x = Dropout(0.5)(x)
5. x = Dense(units=200)(x)
6. x = Activation(actfunc)(x)
7. x = Dropout(0.5)(x)
8. # x = Dense(units=20)(x)
9. # x = Activation(actfunc)(x)
10. # x = Dropout(0.5)(x)
11. x = Dense(units=len(outputs))(x)
12. x = Activation(actfunc)(x)
13.
14. model = Model(inputs=[i1, i2], outputs=x)

```

All activation functions were set as linear mode, which is an identity function. For the loss function, the mean squared error was used. The optimizers used were both the Adam optimizer and the Adadelta optimizer. For metrics, both the mean absolute error and a custom percentage function – defined in

Figure 28, lines 1-2 – were used. The mean square error and the mean absolute error functions can be found on keras documentation, as well as on the equations of **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** also presents the custom percentage function used as a metric.

The architectures were developed, firstly, experimentally. Convpool layers, dense layers, loss functions, optimizers, activation functions, and batch normalization layers were changed until a good performance was achieved. This was when the architecture of network A was found, in Table 4. By experimenting with different convpool layer quantities, networks B and C were created. After choosing the neural network with the best performance, its dense layer quantities were varied, creating the networks D and E. Finally, again by choosing the network superior on performance, the same architecture with a different optimizer was tested, creating the network F. Each network was trained for 200 epochs – a value

found experimentally. Their predictions are gattered and their performance evaluated in section 4.5.3.

Table 3 - Mean squared error and mean absolute error functions.

|                           |   |
|---------------------------|---|
| Mean squared error (mse)  | $mse = average((y_{pred} - y_{true})^2)$                  |
| Mean absolute error (mae) | $mae = average( y_{pred} - y_{true} )$                    |
| Percentage (p)            | $p = average(100 * \frac{y_{pred} - y_{true}}{y_{true}})$ |

Figure 28 - CNN loss, optimizer, and metrics. Source: Author.

```

1. def percentage(y_true, y_pred):
2.     return k.mean(k.abs(100*(y_pred-y_true)/y_true))
3.
4. compileConfig = {
5.     'loss':'mse',
6.     'optimizer':'adam',
7.     'metrics':['mae', percentage],
8.     'percentage':'return k.mean(k.abs(100*(y_pred-y_true)/y_true))'
9. }
10.
11. model.compile(
12.     loss=compileConfig['loss'],
13.     optimizer=compileConfig['optimizer'],
14.     metrics=compileConfig['metrics']
15. )

```

Table 4 – Convolutional Neural Network Architectures.

|                                   | A                     | B                   | C            | D                        | E                        | F                        |
|-----------------------------------|-----------------------|---------------------|--------------|--------------------------|--------------------------|--------------------------|
| Convpool<br>Layers                | 5                     | 4                   | 3            | 5                        | 5                        | 5                        |
| Filters                           | 8, 16, 32,<br>64, 128 | 8, 16,<br>32,<br>64 | 8, 16,<br>32 | 8, 16,<br>32, 64,<br>128 | 8, 16,<br>32, 64,<br>128 | 8, 16,<br>32, 64,<br>128 |
| Hidden units<br>for<br>parameters | 2048                  | 4096                | 8192         | 2048                     | 2048                     | 2048                     |
| Dense layers                      | 1                     | 1                   | 1            | 3                        | 2                        | 1                        |
| Hidden Units                      | 200                   | 200                 | 200          | 500,<br>200, 20          | 200, 20                  | 200                      |
| Optimizer                         | Adam                  | Adam                | Adam         | Adam                     | Adam                     | Adadelta                 |

On this table, filters correspond to the number of feature maps after the convolution operation, not directly the kernel number. The number of hidden units for the parameter branch is the number of neurons on the dense layer of the input parameter branch. Finally, the hidden units' entry stands for the neurons in each of the dense layers after the concatenation of both branches.

### 4.5.3 Convolutional Neural Network Evaluation

For evaluating the neural networks, their predictions must be gathered. The method `get_predictions` on line 1 from Figure 29 is responsible for such.

This method uses the same custom image generator of section 4.5.1. It gets the input ( $x$ ) and target ( $y$ ) values from the generator (`gen`) in Figure 29, line 5. Based on the input, a prediction is made on line 6. Both prediction and target values are stored in variables on lines 7 to 14. The method stops once the number of predictions made reaches the data size, on lines 15 and 16. Predictions are gathered both for the training and the testing data sets.

Figure 29 - Convolutional neural network prediction gathering code.

```

1. def get_predictions(df, path, N):
2.     predictions = np.array([])
3.     targets = np.array([])
4.     gen = generate_generator_multiple(generator=imgen, df=df, path=pa
      th, shuffle=False)
5.     for x, y in gen:
6.         p = model.predict(x)
7.         if predictions.size == 0:
8.             predictions = p
9.         else:
10.            predictions = np.concatenate((predictions, p))
11.        if targets.size == 0:
12.            targets = y
13.        else:
14.            targets = np.concatenate((targets, y))
15.        if len(targets) >= N:
16.            break
17.    return predictions, targets
18.    train_pred, train_targ = get_predictions(train_df, train_path, len(im
      age_files))
19.    test_pred, test_targ = get_predictions(test_df, test_path, len(test_i
      mage_files))

```

An error can be calculated from the prediction and target values. By using the `norm` class from the `scipy` library, this error mean and standard deviation can be calculated and then compared. Besides plotting normal curves, the `plot_error_norm` method on **Erro! Autoreferência de indicador não válida.** calculates those values, on line 7.

Figure 30 - Error mean and standard deviation calculation.

```

1.  from scipy.stats import norm
2.
3.  def plot_error_norm(predictions, targets, title=None):
4.      error = predictions - targets
5.      plt.figure(figsize=(16,6))
6.      plt.xlim(-1,1)
7.      mu, std = norm.fit(error)
8.      t = np.linspace(mu-3*std, mu+3*std, 100)
9.      p = norm.pdf(t, mu, std)
10.     h = plt.plot(t, p, linewidth=2)
11.     plt.grid()
12.     plt.annotate(xy=(mu, max(p)), s="{:.4f} ± {:.4f}".format
(mu, std))
13.     if len(outputs)>1:
14.         plt.legend("Width and Height error distribution")
15.     else:
16.         plt.legend(outputs+" error distribution")
17.     if title:
18.         plt.title(title)
19.     return mu, std

```

The coefficient of determination ( $R^2$ ) from the target versus prediction values was also calculated and used as a mean of comparison between network performance. Those results can be seen in chapter 5.



## 5 Results

This chapter presents the performance results for all developed convolutional neural networks. The training results are discussed in the first section, 5.1, presenting the resulting loss and metrics values. The subsequent section, 5.2, shows the performance of the neural networks on estimating the width and height values. Firstly, the plots of the target versus prediction values are presented, along with their coefficient of determination ( $R^2$ ) values, on section 5.2.1. The  $R^2$  values are then compared for all networks. Later, on section 5.2.2, the error between target and prediction values is evaluated for both training and testing phases and their mean and standard deviation values are discussed.

### 5.1 Training results

The first aspects in which one can evaluate how well did a neural network train is the loss value, along with any further metric values. Watching how fast the loss value decreases throughout the epochs is the measurement of how fast the neural network learns the data.

This learning speed is reflected in the loss curves in Figure 31. Here, all networks have both width and height values as outputs. As explained before (**Erro! Fonte de referência não encontrada.**), the loss value is a mean squared error, calculated by averaging the squared errors between the target and the predicted values. Each data point represents the loss value on the end of each epoch for each convolutional neural network architecture. Starting from the first epoch, the value rapidly decreases on the first 10 epochs. Beginning on around 30 epochs, all networks descend their loss value steadily until the end of the 200 epochs. The fastest learner is the network F, closely followed by network A, both with 5 convolutional layers and a single dense layer with 200 neurons. This difference in speed appears due to the different optimizers for networks A (Adam) and F (Adadelta). The slowest learner is network D, also with 5 convolutional layers but 3 dense layers. A larger network takes longer to train, leading to the result mentioned before.

Figure 31 – Loss value from 200 epochs training of all neural networks with both width and height outputs.

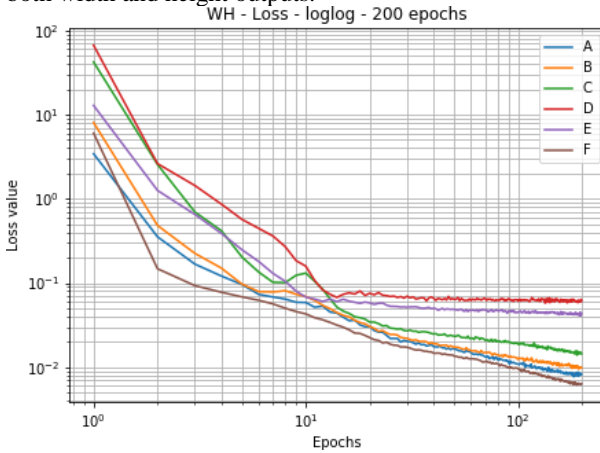
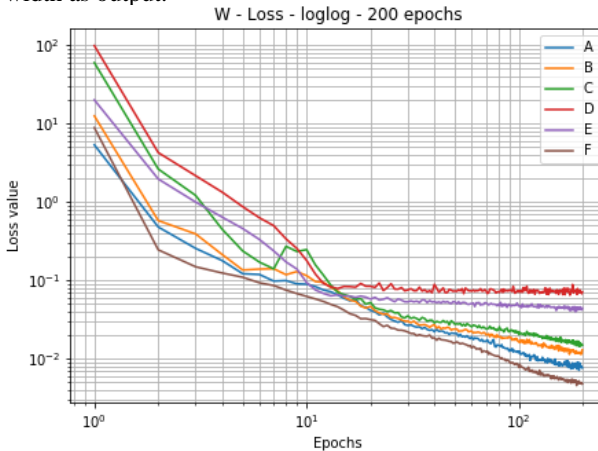
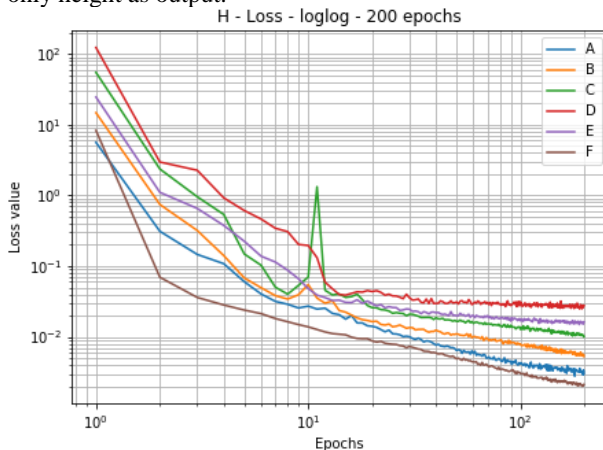


Figure 32 – Loss value from 200 epochs training of all neural networks with width as output.



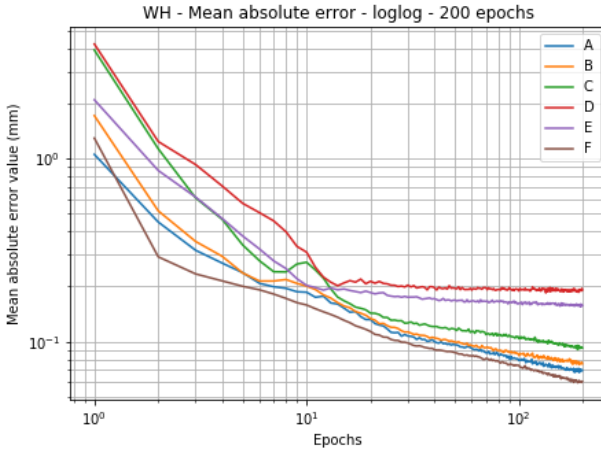
Networks which outputs only one of the width and height dimensions were also trained, their results on Figure 32 (width) and Figure 33 (height). The performances followed the behaviors of the previous case (Figure 31) but most of them achieved lower values than on the multiple output case.

Figure 33 – Loss value from 200 epochs training of all neural networks with only height as output.



Although directly related to the training performance, the loss value does not have a direct physical meaning of the current error levels of the networks. Metrics can help with this visualization. The mean absolute error progression throughout the epochs for the networks with both outputs can be seen in Figure 34. It is similar in shape with the loss graphs, however, it brings an order of magnitude. The network F, for example, has an average error of more than a millimeter on the first epoch. It means that the estimated dimensions are, on the average, more than a millimeter either larger or smaller than their real measurement. This error reduces to less than 0.2 millimeters on epoch 10, and to around 0.06 millimeters by epoch 200.

Figure 34 – Mean absolute error values from 200 epochs training of all neural networks with both width and height as outputs, in millimeters.



As happened with the loss values, the mean absolute error from single output networks presented a similar behavior than from the multiple output ones, on Figure 35 (width) and Figure 36 (height). It can be observed that the networks with height as output achieved slightly lower error values. Again, network F with the height output presented the lowest error value – under 0.03 mm by epoch 200. This is already really close to the values found in the literature (TOYSERKANI; KHAJEPOUR, 2006).

Figure 35 – Mean absolute error values from 200 epochs training of all neural networks with width as output, in millimeters.

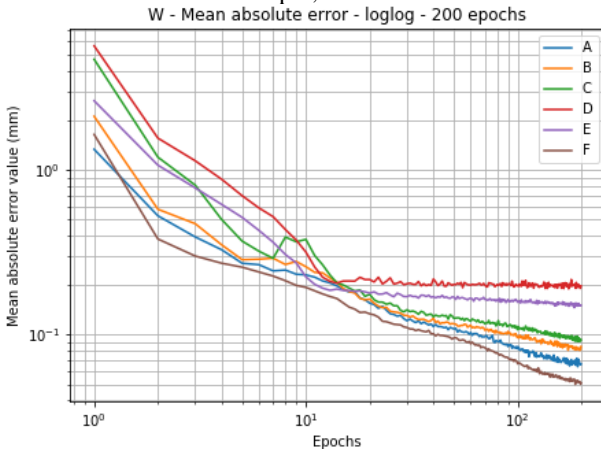
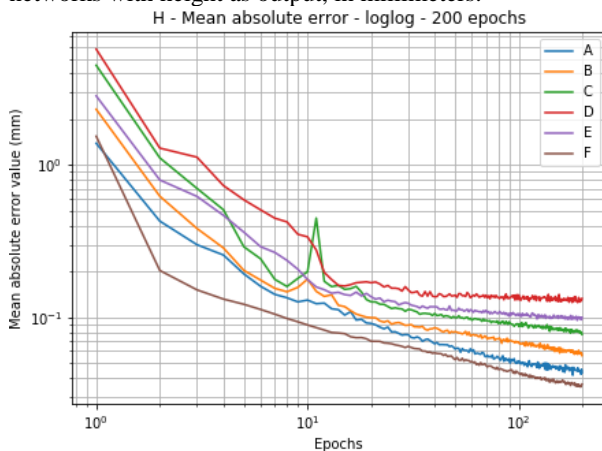


Figure 36 – Mean absolute error values from 200 epochs training of all neural networks with height as output, in millimeters.



This metric does help the visualization of the results of the networks, but still tricks the viewer. The errors on height are smaller because it has smaller values than the width dimension. Because of that, the width error of 0.1 millimeters for a bead 0.5 millimeter tall is way bigger than for a 3 millimeter wide clad bead. The real dimensions of the clad bead should be taken into account, either width or height.

The last metric solves this problem by measuring the average error in proportion to the real dimension, resulting in a percentual error. Its equation was already presented in section 4.5.2 (**Erro! Fonte de referência não encontrada.**). This metric can be seen in Figure 37 for the multiple output networks, and in Figure 38 (width) and Figure 39 (height) for the single output networks.

Figure 37 – Percentage error values from 200 epochs training of all neural networks with both width and height as outputs.

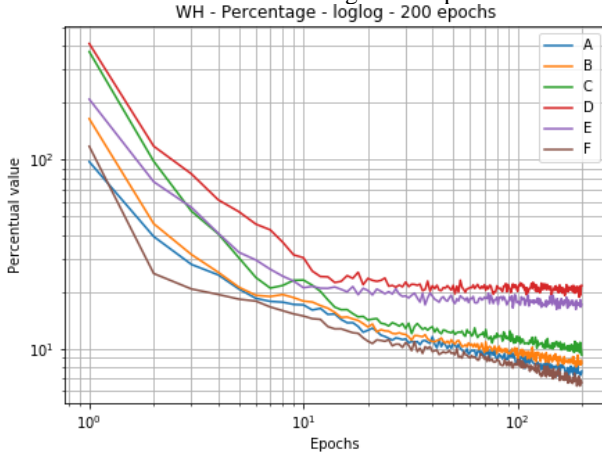


Figure 38 – Percentage error values from 200 epochs training of all neural networks with width as output.

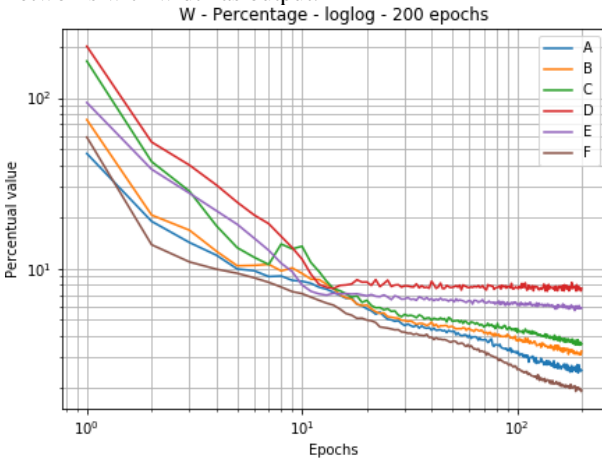
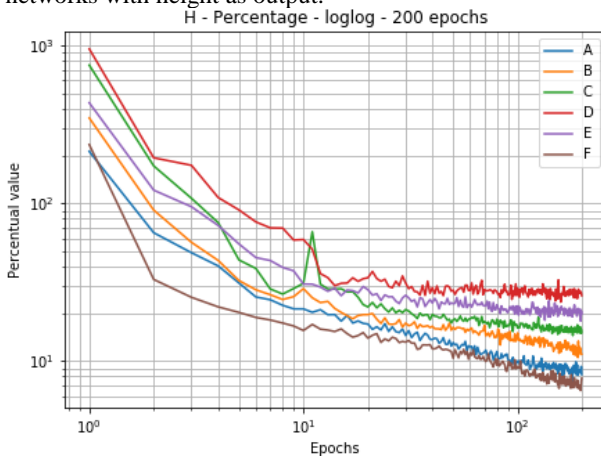


Figure 39 – Percentage error values from 200 epochs training of all neural networks with height as output.



From this metric, it can be observed the opposite behavior. Here, the width dimension percentual error achieves the lowest value among all, below 3% for network F by 100 epochs, while the percentual error of the height dimension remains by around 9% on the same time. The network with multiple outputs stays in between, with around 7%. This proves that the width dimension is easier to learn than the height dimension. This is a logical conclusion, as the width of the molten pool can be seen on the images fed to the networks, although height can not. As explained in chapter 3, molten pool width and clad bead width are in a close relationship (HOFMAN; DE LANGE; MEIJER, 2006). Molten pool height, however, can be only inferred from the size and the brightness of the molten pool.

After analyzing those results, it is expected for the width estimation to have the best performance, followed by both dimensions estimation, and, lastly, by the height estimation. The actual network results are presented in section 5.2.

## 5.2 Estimation performance

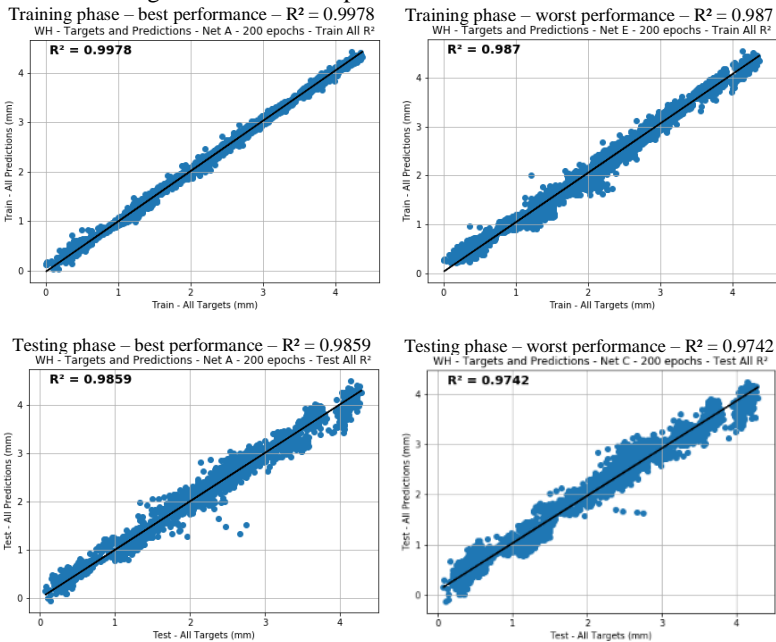
Even better than to analyze the loss value and metrics from the training is to analyze the outputs themselves. As cited before, each one of the six networks was executed with the three different outputs – clad bead width, height and both simultaneously. Their results can be compared to

the target values, which they were supposed to yield, those graphs being on section 5.2.1. Then, the error between both values is analyzed in section 5.2.2.

### 5.2.1 Target and Prediction values

Perhaps the most straightforward way to visualize the networks' performance is to plot their predicted values – the network outputs – versus the target values. To enhance the networks comparability even further, coefficients of determination ( $R^2$ ) for each of those plots were calculated. Regarding this coefficient, the best and worst plots from both training and testing phases can be seen in Figure 40 for the networks with both width and height outputs.

Figure 40 – Best and worst network performances from both testing and training phases. Top row: Training phase. Bottom row: Testing phase. Left column: Best performance. Right column: Worst performance.



Those graphs show the best and the worst performance for the beforementioned outputs. One can notice how higher the  $R^2$  coefficients are on the training phase. It is only natural for it to happen due to the very



training mechanism, where the same images with the same target values are repeated over and over again, throughout the epochs, until the network learns them. On the testing phase, however, it is the first time the network sees each input image, only guessing what the target value could be. The testing performance achieved an  $R^2$  value of 0.986, which is even superior to some of the results presented in the literature (MONDAL; BANDYOPADHYAY; PAL, 2010; HUAMING, 2018).

Another remarkable note is that the training procedure was stopped before a 100% accuracy on the training dataset. It did not make sense to train it until perfection on an imperfect dataset, due to the errors previously mentioned in section 4.3.

The next fact that can be observed is the superiority of network A (5 convpool layers and a single hidden-units layer) when compared to the other networks.

Having more convolutional layers does not mean that network A is the biggest. As seen in Table 4, the C network has the same amount of hidden-units (neurons) as the A network but has 2 convpool layers less than it. After flattening, this leads to 4 times more neurons than the network A, taking way longer to process. One may think that a solution to this is to increase the number of epochs. This, although, could lead to overfitting, which happens when the network memorizes most of the dataset. An overfitted network performs superbly on the training set, yet poorly on the testing set.

The network E has the same amount of convpool layers than the network A but has a single hidden-units dense layer more than it, as seen in Table 4. This extra layer reduced the network's training performance to the point it became the worst, yet closely followed by the performances of networks D and C, as seen in Table 5. The networks C, D, and E are the largest of them, taking longer to train. They could have a better performance if the number of epochs was increased, but networks can not train forever. As mentioned before, increasing the number of epochs could lead to overfitting.

Table 5 –  $R^2$  value for the networks with both width and height outputs.

|             | <b>Width and Height</b> |          |          |          |          |          |
|-------------|-------------------------|----------|----------|----------|----------|----------|
|             | <b>A</b>                | <b>B</b> | <b>C</b> | <b>D</b> | <b>E</b> | <b>F</b> |
| Train $R^2$ | 0.9978                  | 0.9958   | 0.9887   | 0.9876   | 0.9870   | 0.9971   |
| Test $R^2$  | 0.9859                  | 0.9834   | 0.9742   | 0.9831   | 0.9793   | 0.9851   |

Regarding the single output networks, for the width output case, the F network performance outstands. It has the same architecture as network A but was trained with a different optimizer. The worst performances on the training phase were from networks D and E, which are the largest networks as well. On the testing phase, the D network performed way better than it was expected for its size. As the network weights are all initialized at random values, it could be the case that this network started its training with weights closer to the optimum values than the rest of the networks. It still did not outperform the F network. Those performances are shown in Table 6.

Table 6 – R<sup>2</sup> value for the networks with width output.

|                      | <b>Width</b> |          |          |          |          |          |
|----------------------|--------------|----------|----------|----------|----------|----------|
|                      | <b>A</b>     | <b>B</b> | <b>C</b> | <b>D</b> | <b>E</b> | <b>F</b> |
| Train R <sup>2</sup> | 0.9867       | 0.9912   | 0.9836   | 0.9629   | 0.9626   | 0.9954   |
| Test R <sup>2</sup>  | 0.9404       | 0.9369   | 0.9315   | 0.9465   | 0.9336   | 0.9496   |

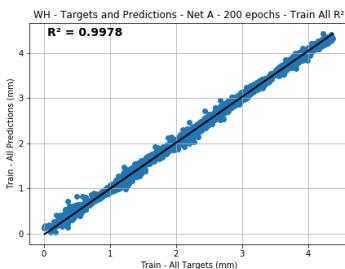
A more consistent performance could be observed from the networks with only the height output. The larger networks had the worst performance in both training and testing phases. For the best performance, there was a tie between networks A and F on the testing phase. Those performances can be seen in Table 7.

Table 7 – R<sup>2</sup> value for the networks with height output.

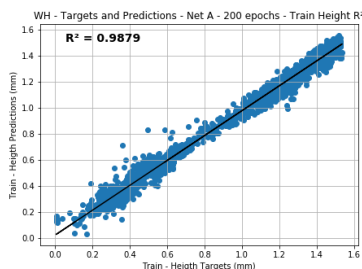
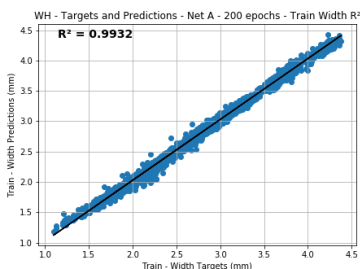
|                      | <b>Height</b> |          |          |          |          |          |
|----------------------|---------------|----------|----------|----------|----------|----------|
|                      | <b>A</b>      | <b>B</b> | <b>C</b> | <b>D</b> | <b>E</b> | <b>F</b> |
| Train R <sup>2</sup> | 0.9947        | 0.9845   | 0.9635   | 0.8823   | 0.9475   | 0.9951   |
| Test R <sup>2</sup>  | 0.9593        | 0.9483   | 0.9363   | 0.8491   | 0.9193   | 0.9593   |

Figure 41 -  $R^2$  for the multiple output networks, calculated for both outputs (top), only for width (bottom left) and only for height (bottom right).

$R^2 = 0.9978$  for both width and height values



$R^2 = 0.9932$  for only the width value  $R^2 = 0.9879$  for only the height value



When comparing both results, one can infer that the networks with single outputs performed better than the multiple output networks. To do so, one must compare not the  $R^2$  presented in Table 5, which consists of both width and height values, but splitting them both into two different plots for each variable, as seen in Figure 41.

From those values, a fair comparison can be made. On Table 8, it can be observed the  $R^2$  values for both the networks with multiple outputs – calculated individually for each dimension – and the ones with a single output. On average, the single output networks achieved a slightly higher  $R^2$  value than the multiple output ones, with some exceptions. This is a logical outcome, once a neural network with a single output can specialize itself over it, instead of performing averagely for both outputs. Regarding architectures, the networks A and F keep on the lead of performance.

Table 8 – Individual height and width  $R^2$  for all networks.

|   | <b>Multiple Output <math>R^2</math></b> |               |              |               | <b>Single Output <math>R^2</math></b> |               |              |               |
|---|---|---------------|--------------|---------------|---------------------------------------|---------------|--------------|---------------|
|   | <b>Train</b>                            |               | <b>Test</b>  |               | <b>Train</b>                          |               | <b>Test</b>  |               |
|   | <b>Width</b>                            | <b>Height</b> | <b>Width</b> | <b>Height</b> | <b>Width</b>                          | <b>Height</b> | <b>Width</b> | <b>Height</b> |
| A | 0.9932                                  | 0.9879        | 0.9445       | 0.9558        | 0.9867                                | 0.9947        | 0.9404       | 0.9593        |
| B | 0.9836                                  | 0.9841        | 0.9366       | 0.9429        | 0.9912                                | 0.9845        | 0.9369       | 0.9483        |
| C | 0.9752                                  | 0.9108        | 0.9218       | 0.8594        | 0.9836                                | 0.9635        | 0.9315       | 0.9363        |
| D | 0.9587                                  | 0.9367        | 0.941        | 0.9279        | 0.9629                                | 0.8823        | 0.9465       | 0.8491        |
| E | 0.9633                                  | 0.9183        | 0.9319       | 0.9007        | 0.9626                                | 0.9475        | 0.9336       | 0.9193        |
| F | 0.9888                                  | 0.9886        | 0.9404       | 0.9549        | 0.9954                                | 0.9951        | 0.9496       | 0.9593        |

The  $R^2$  value can express well how each neural network performed, although it is not the only possible way to do so. By subtracting the target values from the estimated values, an estimation error is calculated. An analysis of this error value is in section 5.2.2.

### 5.2.2 Estimation error

The difference between the estimation and the target values is the estimation error, yet another way to evaluate a neural network's accuracy. By considering the data to be normally distributed, this error can be fit into a Gaussian distribution, thus, a mean and a standard deviation can be calculated. Those gaussian distributions for the networks with multiple outputs can be seen in Figure 42, the left graph standing for the training phase and the right, for the testing phase.

The first difference to be observed on those graphs is their magnitudes. The networks A, B, and F have peaked on their training phases, thus, a lower standard deviation, as seen in Table 9. On their testing phases, however, they perform similarly to the remaining networks. This is a clear indication of overfitting.

Those networks have memorized the training dataset. Usually, this means the network would perform poorly on the testing phase, even so, they are still the top performance networks. One viable explanation is the similarity of the images between training and testing data sets. Perhaps a better approach on dataset splitting, instead of splitting them experiment-wise, would be to merge all of the data and let the built-in keras algorithm split a percentage of the dataset into testing data.

The remaining networks performed slightly worst on their testing phase, as expected due to the learning mechanism. Regarding the mean

value, all values remained below 0.1 mm, which indicate a mean value tending to zero, as expected on a normal distribution.

Figure 42 – Gaussian distributions of the estimation error for the networks with multiple outputs on their training phase (left) and testing phase (right).

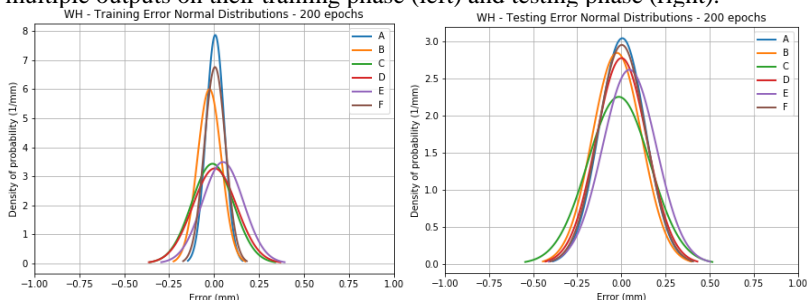


Table 9 – Mean and standard deviations for the estimation error on both training and testing phases for the multiple output networks.

| <b>Width and Height Error (<math>\mu\text{m}</math>)</b> |              |                           |             |                           |
|--|--------------|---------------------------|-------------|---------------------------|
|  | <b>Train</b> |                           | <b>Test</b> |                           |
|  | <b>Mean</b>  | <b>Standard deviation</b> | <b>Mean</b> | <b>Standard deviation</b> |
| A  | 6.56         | 50.72                     | 5.29        | 130.85                    |
| B  | -25.65       | 66.50                     | -24.95      | 139.93                    |
| C  | -8.54        | 116.28                    | -14.52      | 176.80                    |
| D  | 3.50         | 122.04                    | 0.56        | 143.47                    |
| E  | 49.49        | 114.37                    | 45.94       | 152.08                    |
| F  | 5.16         | 59.00                     | 2.42        | 134.84                    |

For the width output networks, a similar behavior happens. The F network outstands on its training phase. On the testing phase, the network D falsely surpasses network F due to its luckiest initial set of weights. Despite that, networks A and F remain on the lead. This behavior can be observed in Figure 43. The mean values stay under 0.1 mm, again indicating that this value tends to zero, thus, the data is normally distributed.

On those graphs, the standard deviation values are lower on their training phase than for the multiple output networks, but slightly higher on the testing phase, as seen in Table 10. It implies the presence of even more overfitting than before. It relates to the fact that the network only

needs to learn one output, not both. It is then easier for it to memorize the data.

Figure 43 – Gaussian distributions of the estimation error for the networks with the width output on their training phase (left) and testing phase (right).

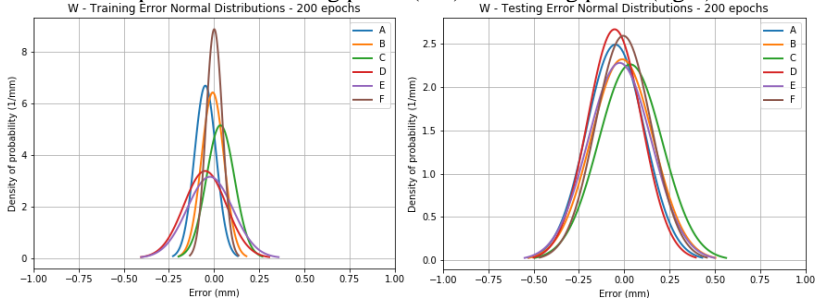


Table 10 – Mean and standard deviations for the estimation error on both training and testing phases for the width output networks.

| <b>Width Error (<math>\mu\text{m}</math>)</b> |              |                           |             |                           |
|---|--------------|---------------------------|-------------|---------------------------|
|   | <b>Train</b> |                           | <b>Test</b> |                           |
|   | <b>Mean</b>  | <b>Standard deviation</b> | <b>Mean</b> | <b>Standard deviation</b> |
| A   | -48.27       | 59.74                     | -49.14      | 159.98                    |
| B   | -7.37        | 62.20                     | -11.80      | 171.71                    |
| C   | 35.39        | 77.56                     | 33.844      | 176.19                    |
| D   | -49.31       | 118.33                    | -53.07      | 149.41                    |
| E   | -22.85       | 126.78                    | -25.32      | 174.85                    |
| F   | 1.12         | 45.00                     | -5.10       | 153.75                    |

Finally, for the height output case, even more overfitting occurs. The F network presents the highest peak – thus the lowest standard deviation – but only on the training phase. This behavior can be observed in Figure 44. Opposing network F, the D network presents the lowest peak, which is the highest standard deviation. The standard deviation value from the network F is the lowest among all training graphs, while the standard deviation value from the D network is the highest among all – those values in Table 11. It means that this output was the easiest to memorize for network F, but the hardest to learn for network D.

Figure 44 – Gaussian distributions of the estimation error for the networks with the height output on their training phase (left) and testing phase (right).

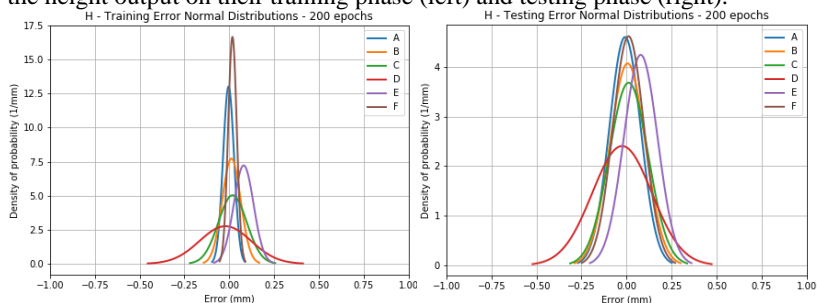


Table 11 – Mean and standard deviations for the estimation error on both training and testing phases for the height output networks.

| <b>Height Error (<math>\mu\text{m}</math>)</b> |              |                           |             |                           |
|--|--------------|---------------------------|-------------|---------------------------|
|  | <b>Train</b> |                           | <b>Test</b> |                           |
|  | <b>Mean</b>  | <b>Standard deviation</b> | <b>Mean</b> | <b>Standard deviation</b> |
| A  | -4.01        | 30.66                     | -8.38       | 86.67                     |
| B  | 12.01        | 51.65                     | 6.03        | 97.97                     |
| C  | 18.43        | 79.25                     | 11.84       | 108.30                    |
| D  | -21.86       | 144.39                    | -24.70      | 165.81                    |
| E  | 80.36        | 55.24                     | 78.73       | 94.01                     |
| F  | 17.678       | 23.97                     | 10.91       | 86.35                     |

Regarding the mean values, again, they tend to zero, indicating the normal distribution of the data. Even with strong overfitting, the networks with more convpool layers and less densely connected layers (A and F) yielded the best results, for all output types. On the present case, the overfit did not degrade the testing dataset measurements, because of its similarity to the training dataset. When processing with different input parameters than the ones here presented, however, those networks' performances will need re-evaluation.

As it was observed, the data fits into normal distributions. When observing only the test phases of networks A and F, one obtains the information in Table 12. The lowest standard deviation value, 86  $\mu\text{m}$ , is from network F, with only the height output. This value indicates that 95% of the measurements have an error under 172  $\mu\text{m}$ . Although it already represents a small error, it is important to remember what this error means.

The networks estimate the width and the height dimensions for each frame. When averaging this error along the length of the clad bead, it

shrinks considerably. The error means in Table 12 reflect this effect, although those are not the values of each individual clad bead. When analyzing the averages, the maximum average error value shrinks to 49  $\mu\text{m}$ , which are comparable to the literature (DAVIS; SHIN, 2011). It also implies a better performance for slower beads, as there would be acquired more frames per unit length.

Table 12 – Error mean and standard deviation of the test phases from networks A and F for all output types.

|   | Width and Height Error ( $\mu\text{m}$ ) |                    | Width Error ( $\mu\text{m}$ ) |                    | Height Error ( $\mu\text{m}$ ) |                    |
|---|--|--------------------|-------------------------------|--------------------|--------------------------------|--------------------|
|   | Mean                                     | Standard deviation | Mean                          | Standard deviation | Mean                           | Standard deviation |
| A | 5.29                                     | 130.85             | -49.14                        | 159.98             | -8.38                          | 86.67              |
| F | 2.4                                      | 134.84             | -5.10                         | 153.75             | 10.91                          | 86.35              |

Apart from that, the dimensions of the clad beads vary significantly. A standard deviation of 130  $\mu\text{m}$  may be seen too large for a bead 1 mm wide although it is not that big for one which is 4.5 mm wide. The best way to compensate for the bead dimensions is to calculate the error as a percentage of the bead's dimensions, which can be simply made by dividing each error value by the corresponding target value, as shown in Table 13.

Table 13 - Percentual error definition.

|                  |   |
|------------------|---|
| Percentual error | $\frac{y_{estimated} - y_{target}}{y_{target}}$ |
|------------------|---|

By calculating the errors from the A and F networks in a similar fashion that in Table 12, one obtains Table 14. There, error average remains below 3%, way lower than some works found in literature, (IRAVANI-TABRIZIPOUR; ASSELIN; TOYSERKANI, 2006; IRAVANI-TABRIZIPOUR; TOYSERKANI, 2007), and with fewer resources. The width dimension appears as the dimension with the lowest standard deviation, which means it is the most stable, in agreement with many authors in the literature (HOFMAN; DE LANGE; MEIJER, 2006; MONDAL; BANDYOPADHYAY; PAL, 2010; ARIAS *et al.*, 2014; MORALEJO *et al.*, 2017). It is logical for width to be the most stable dimension, once it is observable in the molten pool image, what is in agreement with the conclusions in section 5.1. The networks which



outputted both dimensions simultaneously have the highest standard deviation, which is also expected, as the networks could not specialize on both dimensions as well as they did to only one of them, which is again in agreement with the literature (HUAMING, 2018).

Table 14 – Percentual error mean and percentual error standard deviation of the test phases from networks A and F for all output types.

| <b>Test Phase Percentual Error (%)</b> |                           |              |                           |               |                           |       |
|--|---------------------------|--------------|---------------------------|---------------|---------------------------|-------|
| <b>Width and Height</b>                |                           | <b>Width</b> |                           | <b>Height</b> |                           |       |
| <b>Mean</b>                            | <b>Standard deviation</b> | <b>Mean</b>  | <b>Standard deviation</b> | <b>Mean</b>   | <b>Standard deviation</b> |       |
| A                                      | 0.96                      | 26.57        | 2.47                      | 7.55          | 1.71                      | 18.34 |
| F                                      | 2.66                      | 23.05        | 0.20                      | 6.63          | -1.59                     | 13.74 |

Finally, after analyzing all of the obtained results, one can infer if this is a viable method of process monitoring, along with its advantages and disadvantages. This method estimated all of the dimensions with rather an accuracy, equating itself with the results obtained in the literature. It is noticeable how little hardware was needed when comparing it to the previous molten pool image-based approaches, especially the ones on the early days of the technology.

The next chapter is dedicated to further conclusions of this work.



## 6 Conclusion

This work consisted of an implementation of a monitoring system for a laser processing unit. The first step taken for its execution was to build a background on the main fields of the project, thus, leading to the research lines presented in chapter 2. After this, systematic research was performed aiming to find applications of convolutional neural networks on laser cladding process monitoring, however, it was unsuccessful. Many other works related to the area were presented in chapter 3 but none corresponded exactly to the research line. With the researches concluded, the implementation process started. From the hardware choice to the libraries used on each CNN architecture development, everything was detailed in chapter 4. After execution, the results were presented in chapter 5, where it was observed which architectures had a better performance for the acquired dataset. Those were the architectures with the most convpool layers and the least densely connected layers. Although strong overfitting was observed among those networks, it did not demote their performances on their testing phases, keeping their performance on the lead.

The proposed system and the developed CNN architectures were able to satisfactorily estimate the clad bead geometries. The best coefficient of determination values prevailed over 0.99 for the training dataset and over 0.95 for the testing one. After the error analysis, the same networks remained as the most accurate ones, with the least error mean and standard deviation values.

What did not work as expected was the physical setup. A magnification system was designed and manufactured to increase the size of the molten pool on the image acquired by the camera. Due to complications in the manufacturing process and flaws on the design, the system did not operate as expected, leading to blurry images. It was later observed how disposable this system was, as the molten pool image did not need magnification as it was even shrunk down to be fed to the CNN.

Another aspect that could be further explored was the CNN architectures themselves. There were many other parameters on their structures that could have been explored such as the number of hidden units – especially on the input parameter branch – and the filter numbers on the convolutional branch. On the training procedure, there were the loss function, optimizers, the batch size, and larger epoch number, that could also have been experimented with.

There are many other suggestions for future works to be studied below:

- To better match cross sections and frames: the approach here taken introduced errors on the data as the cross sections were not acquired simultaneously to the frames. A laser triangulation system is suggested for an in-process geometry measurement.
- To exclude the lowest power clads: There is no need to work with clad beads that detach themselves from the substrate. On this work, they were not discarded because it would be left way too little data for a network to be properly trained.
- To try for other parameter ranges: The most data, the better it is for network training. It is highly recommended to invest in data acquiring, trying to diversify the input parameter combinations as much as possible.
- To try other material compositions: In the laboratory, no research has been conducted yet neither on different compositions of feedstock nor substrate material and how they interfere on the clad bead geometry
- To reduce the number of neurons for input parameters: The neuron number used for the input parameter is way larger than it is necessary, although this parameter was not explored in this work.
- To increase the number of epochs for all networks: All networks here presented were trained for 200 epochs. There is always the possibility to train further.
- To try for genetic algorithms for achieving optimal network architecture: As suggested in the literature review (HUAMING, 2018), an approach based on genetic algorithms could find the most optimized architecture for this problem.
- A better training/testing dataset splitting: To separate an entire 24 clad beads with unique parameters for the testing data set resulted on this set being really similar to the training one. A better approach would be to let the algorithm randomly decide which ones are used for the testing dataset, controlling only the percentage of the whole dataset to be used as testing.
- To use this CNN to train another CNN which takes images and a desirable clad bead geometry as input and outputs the process input parameters. This second CNN could be a strong tool for a closed loop control of the process.

- To evaluate CNN real-time implantation viability: It is known that neural networks are slow algorithms. Once trained, however, they may be fast enough to even be used as a control algorithm for such a process. This is a huge field of interest for the future of this application.

The overall result of this work is very positive for the laboratory. Being its first ever project on this area, it opens paths of the process monitoring field that are yet to be explored, leading to many new research lines. For the scientific community, it represents yet another alternative to the intelligent process monitoring field.



## 7 Bibliography

AGGARWAL, K.; URBANIC, R. J.; SAQIB, S. M. Development of predictive models for effective process parameter selection for single and overlapping laser clad bead geometry. **Rapid Prototyping Journal**, v. 24, n. 1, p. 214-228, 2018.

ARIAS, J. L. et al. Real-time laser cladding control with variable spot size. **Laser 3D Manufacturing**, 2014. San Francisco, CA. SPIE.

BARUA, S.; SPARKS, T.; LIOU, F. Development of low-cost imaging system for laser metal deposition processes. **Rapid Prototyping Journal**, v. 17, n. 3, p. 203-210, 2011.

CAIAZZO, F.; CAGGIANO, A. Laser direct metal deposition of 2024 Al alloy: Trace geometry prediction via machine learning. **Materials**, v. 11, n. 3, 2018.

DAVIS, T. A.; SHIN, Y. C. Vision-based clad height measurement. **Machine Vision and Applications**, v. 22, n. 1, p. 129-136, 2011.

DOUBENSKAIA, M. et al. Definition of brightness temperature and restoration of true temperature in laser cladding using infrared camera. **Surface and Coatings Technology**, v. 220, p. 244-247, 2013.

DOUKKALI, F. **Batch normalization in Neural Networks**. 2017. Disponível em: < <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c> >. Acesso em: Feb 15.

FERENHOF, H.; FERNANDES, R. **Passo-a-passo para construção da Revisão Sistemática e Bibliometria Utilizando a ferramenta Endnote®**. 3.06 2018.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in

position. **Biological Cybernetics**, v. 36, n. 4, p. 193-202, 1980/04/011980.

HOFMAN, J. T.; DE LANGE, D. F.; MEIJER, J. Camera based feedback control of the laser cladding process. **ICALEO 2006 - 25th International Congress on Applications of Laser and Electro-Optics**, 2006. Scottsdale, AZ.

HU, D.; KOVACEVIC, R. Modelling and measuring the thermal behaviour of the molten pool in closed-loop controlled laser-based additive manufacturing. **Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture**, v. 217, n. 4, p. 441-452, 2003.

HU, D.; MEI, H.; KOVACEVIC, R. Improving solid freeform fabrication by laser-based additive manufacturing. **Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture**, v. 216, n. 9, p. 1253-1264, 2002.

HUAMING, L. X., QIN; SONG, HUANG; LEI, JIN; YONGLIANG, WANG; KAIYUN, LEI. Geometry Characteristics Prediction of Single Track Cladding Deposited by High Power Diode Laser Based on Genetic Algorithm and Neural Network. **International Journal of Precision Engineering and Manufacturing**, v. 19, n. 7, p. 1061-1070, 2018.

IMAGINGSOURCE, T. **DFG/USB2pro Video-to-USB 2.0 converter**. Disponível em: < <https://www.theimagingsource.com/products/converters-grabbers/video-to-usb-2.0-converters/dfgusb2pro/> >. Acesso em: 05/04/2018.

INC., L. P. **Deep Learning: Convolutional Neural Networks in Python**. 2018. Disponível em: < <https://www.udemy.com/deep-learning-convolutional-neural-networks-theano-tensorflow/> >.



IRAVANI-TABRIZIPOUR, M.; ASSELIN, M.; TOYSERKANI, E. Development of an image-based feature tracking algorithm for real-time clad height detection. **4th IFAC Symposium on Mechatronic Systems, MX 2006**, 2006. Heidelberg. PART 1. p.914-920.

IRAVANI-TABRIZIPOUR, M.; TOYSERKANI, E. An image-based feature tracking algorithm for real-time measurement of clad height. **Machine Vision and Applications**, v. 18, n. 6, p. 343-354, 2007.

KAPPA. **CF 8/5 MX and DSP**. 2006. Disponível em: < <https://www.cm-tech.at/upload/8598335-3583663-CF-8-5-MX-E.pdf> >. Acesso em: 05/04/2018.

KERAS. **Image Preprocessing**. 2018. Disponível em: < <https://keras.io/preprocessing/image/> >. Acesso em: 2018.

LE CUN, Y. **Generalization and network design strategies**. 1989.

LEI, J.; WANG, Z.; LIU, L. **Design of forming shape measurement system for laser molten pool in laser fabricating**. International Conference on Engineering Design and Optimization, ICEDO 2010. Ningbo. 37-38: 327-330 p. 2010.

LIU, J.; WU, Y.; WANG, L. In-situ measurement based on prior calibration with analogist samples for laser cladding. **High-Power Lasers and Applications VI, November 5, 2012 - November 5, 2012**, 2012. Beijing, China. SPIE. p.The Society of Photo-Optical Instrumentation Engineers (SPIE); Chinese Optical Society (COS).

MERIAUDEAU, F.; RENIER, E.; TRUCHETET, F. CCD technology applied to laser cladding. In: ANAGNOSTOPOULOS CONSTANTINE, N.; BLOUKE MORLEY, M., *et al*, **Solid State Sensor Arrays and CCD Cameras**, 1996. San Jose, CA, USA. p.299-309.

MERIAUDEAU, F.; TRUCHETET, F. Control and optimization of the laser cladding process using matrix cameras and image processing. **Journal of Laser Applications**, v. 8, n. 6, p. 317-324, 1996.

MERIAUDEAU, F.; TRUCHETET, F. Image processing applied to the laser cladding process. **High-Power Lasers: Applications and Emerging Applications**, v. 2789, p. 93-103, 1996.

MERIAUDEAU, F. et al. Acquisition and image processing system able to optimize laser cladding process. **Proceedings of the 1996 3rd International Conference on Signal Processing, ICSP'96. Part 1 (of 2)**, 1996. Piscataway, NJ, United States  
Beijing, China. IEEE. p.1628-1631.

MONDAL, S.; BANDYOPADHYAY, A.; PAL, P. K. An experimental investigation into the optimal processing conditions for the co2 laser cladding of 20 MnCr5 steel using taguchi method and ANN. **International Conference on Modeling, Optimization, and Computing, ICMOC 2010**, 2010. Durgapur, West Bengal. p.392-398.

MORALEJO, S. et al. A feedforward controller for tuning laser cladding melt pool geometry in real time. **International Journal of Advanced Manufacturing Technology**, v. 89, n. 1-4, p. 821-831, 2017.

OCYLOK, S. et al. Correlations of melt pool geometry and process parameters during laser metal deposition by coaxial process monitoring. In: SCHMIDT, M.; MERKLEIN, M., *et al*, **International Conference on Laser Assisted Net Shape Engineering, LANE 2014**, 2014. C: Elsevier B.V. p.228-238.

TOYSERKANI, E. **Laser Cladding**. 2005.

TOYSERKANI, E.; KHAJEPOUR, A. A mechatronics approach to laser powder deposition process. **Mechatronics**, v. 16, n. 10, p. 631-641, Dec2006.

VIJAYABHASKAR, J. **Tutorial on Keras ImageDataGenerator with flow\_from\_dataframe.** 2018. Disponível em: <  
<https://medium.com/@vijayabhaskar96/tutorial-on-keras-imagedatagenerator-with-flow-from-dataframe-8bd5776e45c1>>.

XING, F.; LIU, W.; WANG, T. Real-time sensing and control of metal powder laser forming. **6th World Congress on Intelligent Control and Automation, WCICA 2006**, 2006. Dalian. p.6661-6665.