 doi:10.20944/preprints202108.0272.v1

Article

Remaining Useful Life Prediction from 3D Scan Data with Genetically Optimized Convolutional Neural Networks

Giovanni Diraco [1*], Pietro Siciliano[2], and Alessandro Leone[3]

[1] National Research Council of Italy, IMM—Institute for Microelectronics and Microsystems, 73100 Lecce, Italy; giovanni.diraco@cnr.it
[2] National Research Council of Italy, IMM—Institute for Microelectronics and Microsystems, 73100 Lecce, Italy; pietro.siciliano@le.imm.cnr.it
[3] National Research Council of Italy, IMM—Institute for Microelectronics and Microsystems, 73100 Lecce, Italy, alessandro.leone@cnr.it
* Correspondence: giovanni.diraco@cnr.it; Tel.: +39 832 422 531

Abstract: In the current industrial landscape, increasingly pervaded by technological innovations, the adoption of optimized strategies for asset management is becoming a critical key success factor. Among the various strategies available, the "Prognostics and Health Management" strategy is able to support maintenance management decisions more accurately, through continuous monitoring of equipment health and "Remaining Useful Life" forecasting. In the present study, Convolutional Neural Network-based Deep Neural Network techniques are investigated for the Remaining Useful Life prediction of a punch tool, whose degradation is caused by working surface deformations during the machining process. Surface deformation is determined using a 3D scanning sensor capable of returning point clouds with micrometric accuracy during the operation of the punching machine, avoiding both downtime and human intervention. The 3D point clouds thus obtained are transformed into bidimensional image-type maps, i.e., maps of depths and normal vectors, to fully exploit the potential of convolutional neural networks for extracting features. Such maps are then processed by comparing 15 genetically optimized architectures with the transfer learning of 19 pretrained models, using a classic machine learning approach, i.e., Support Vector Regression, as a benchmark. The achieved results clearly show that, in this specific case, optimized architectures provide performance far superior (MAPE=0.058) to that of transfer learning which, instead, remains at a lower or slightly higher level (MAPE=0.416) than Support Vector Regression (MAPE=0.857).

Keywords: Remaining Useful Life; Deep Neural Network; Convolutional Neural Network; Genetic Optimization; Neural Network Optimization; Support Vector Regression; Depth Maps; Normal Maps; 3D Point Clouds.

1. Introduction

Over recent years, asset maintenance has received increasing attention in the literature. If considering that appropriate maintenance management has a direct effect on reducing costs and increasing system reliability, availability, and safety [1], it is easy to understand how asset maintenance is a critical success factor in the current industrial revolution 4.0 [2]. More specifically, the adoption of optimized maintenance strategies has proven to improve tool utilization and productivity, assuring product quality and operational excellence [3].

With the main objective of reducing unexpected breakdowns and possibly catastrophic consequences, maintenance strategies can be roughly classified under [4]: 1) corrective maintenance, 2) preventive maintenance, and 3) prognostics and health management (PHM). In the case of corrective maintenance, machine tools are operated until the tool breaks down, and repairs are made at the time of failure. However, on the other hand, if a critical breakdown occurs, it may cause serious machinery damages. The preventive maintenance aims to prevent the aforementioned problems by scheduling inspection and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

repair interventions at regular time intervals or operation cycles. In this case, the bigger downside is the waste of time and the replacement costs for components that often are still working. Contrary to previous strategies, PHM relies on the continuous monitoring of equipment health conditions to predict the degradation status, in terms of remaining useful life (RUL), supporting thus more accurate maintenance management decisions.

The prediction of RUL, definable as the "length from the current time to the end of the useful life" as suggested in [5], can be achieved by at least three types of approaches [6]: model-based, data-driven, and hybrid. The model-based approaches (also known as physics-based) refer to mathematical formulations able to model the physical degradation process for the purpose of estimating RUL. In the case of data-driven approaches, instead, RUL is estimated from degradation data collected by monitoring sensors, and processed using traditional statistical or machine learning (ML) techniques or even more "advanced" ones based on deep neural networks (DNNs) [7]. However, very often the choice of the appropriate approach depends on the specific problem at hand. Thus, aiming to exploit the strengths of both approaches, data-driven and model-based, they are combined in the hybrid approaches by using some kind of fusion scheme [6].

Exploiting the laws of nature to model system degradation, model-based approaches are generally quite accurate. Nevertheless, the implementation of a faithful model for predicting RUL is an expensive and time-consuming process; it may be feasible for simple parts, but may not be for complex components or systems due to the limited understanding of their behavior under all operating conditions. Such disadvantages combined with the risk of not achieving the desired results, make model-based approaches definitely less attractive than data-driven ones [8].

The various data-driven methods revolve around the processing of features obtained from monitoring sensor data. Consequently, the most important distinction differentiating among such methods lies in the way features are obtained, that is, either by traditional handcrafted methods or by learned representations. The methods belonging to the first category utilize representative features, extracted and selected by hand (i.e., handcrafted) on the basis of expert domain knowledge, and then classified or evaluated via regression using appropriate statistical or ML techniques.

The disadvantages of these methods are that handcrafted features are representative only of a specific component or system under certain conditions, while, on the other hand, the process of feature extraction and selection is time-consuming and laborious, relying often on strong prior domain knowledge [9]. Moreover, as shown by [10], the performance of ML algorithms is limited by data representation.

In the case of learned representations, on the contrary, representative features of degradation states can be automatically discovered from sensor data by using Deep Learning (DL) techniques [7, 11]. Up to now, a lot of fruitful research results involving many different fields, ranging from image recognition to natural language processing, have been reported in the DL literature [12]. Although, recently, an increasing number of research studies exploiting DL has appeared in the RUL literature, there is still less availability of optimized DL models and architectures compared to other fields.

In the present study, a new DL model based on convolutional neural network (CNN) is proposed for the RUL prediction of punching tools. The main contributions are (i) representation of punch deformation with depth and normal vector maps (DNVMs) obtained from 3D scan point clouds; (ii) CNN-based RUL prediction with network architecture optimized using genetic algorithm (GA); (iii) validation of the proposed method with real data sets representing three different deformation modes.

The remaining of this paper is organized as follows: Section 2 covers a brief literature review of DL-based RUL prediction; Section 3 presents the materials and methods adopted for design, implementation and experimental validation; the empirical results are provided in Section 4 and detailed discussed on Section 5; finally, some conclusions are drawn in Section 6.

3. Related works

 doi:10.20944/preprints202108.0272.v1

Complex real-world data are very useful in many machine-learning applications, including RUL prediction, but they are also cumbersome to process, transmit and store, due to their high-dimensional nature. More effective and low-dimensional features can be obtained from high-dimensional data by using representation learning techniques. The year 2006 marked an important turning point in this research area, since earlier widely-used methods such as principal component analysis (PCA) [13, 14] and linear discriminant analysis (LDA) [15] have given way to more advanced DL methods [16]. DL architectures, in contrast to shallow ones, are composed by multiple data transformation layers, providing higher hierarchical abstraction levels and thus more useful for classification, detection and prediction tasks.

One of the most common DL approach is the stacked sparse auto-encoder (SAE), in which a network of multiple encoder layers is used to transform high-dimensional data into low-dimensional features and, conversely, a network of multiple decoder layers recovers back the original data. Specifically, the RUL of an aircraft engine was predicted by Ma et al. [17] by using SAE to extract performance degradation features. Sun et al. [18] addressed the problem of deep transfer learning with SAE networks for predicting RUL of cutting tool. They investigated three different transfer strategies, i.e., weight transfer, feature transfer, and weight update, to transfer a trained SAE to a new object tool under operation without providing supervised training information. The authors claimed that deep transfer learning improves performance of RUL prediction also in case of few historical failure data for training. Ren at al. [19] proposed a DL-based framework for bearing RUL prediction using deep auto-encoder and time-frequency-wavelet joint features to representing the bearing degradation process. As the authors pointed out, the advantages offered by the deep autoencoder method were twofold, i.e., automatic feature selection and over fitting problem prevention thanks to reducing network parameters.

Another neural network class arousing considerable research interest in feature learning is represented by the Restricted Boltzmann Machine (RBM). It is an energy-based neural network with two layers of stochastic binary neurons, one is the visible layer and the other one is the hidden layer. The main issues when dealing with RBM (even stacked in multiple layers) is the model parameter initialization (e.g, learning rate, momentum, number of hidden units, mini-batch size , etc.) and how to regularize the model to avoid over fitting and improve the learning process. Liao et al. [20] addressed the regularization problem by suggesting a new term allowing to train an RBM to output a feature space that better represents degradation patterns in RUL prediction. Although one RBM layer was used, they pointed out that their method can be extended by stacking multiple RBM layers in a deeper neural network architecture.

Haris et al. [21] addressed the problem of find optimal hyperparameters for a Deep Belief Network (DBN), which is a generative model composed of multiple RMB layers, at the purpose to predict the RUL of supercapacitors. To this end, they proposed a combination of Bayesian and HyperBand optimization, and showed the universality of their model by training it on different degradation profiles with the same hyperparameters.

Aiming to predict RUL of complex engineering system whose malfunctions may be caused by multiple faults, Jiao et al. [22] proposed a RUL prediction framework for multiple fault modes consisting of three main modules: DBN-based extraction of degradation features where original data were preprocessed with a gap metric; fault identification under multiple fault conditions by support vector data description (SVDD) monitoring; RUL estimation via particle filter (PF) and adaptive failure threshold.

Ma et al. [23] assessed the health condition of a bearing rig by using a discriminative DBN model composed of four layers. They obtained the model parameters, i.e., the number of neurons of the two hidden layers and the learning rate, by using the ant colony optimization (ACO) algorithm. The data consisted of vibration signals collected at 10 min intervals with a sampling frequency of 10 kHz. The degradation prediction was formulated as a classification problem with five classes representing the bearing conditions during the evolution process.

 doi:10.20944/preprints202108.0272.v1

Zhang et al. [24] proposed a DBN-based ensemble method for RUL prediction in which multiple DBNs were evolved using a multi-objective evolutionary algorithm integrated with the traditional DBN training technique. They used DBNs with three hidden layers, whose optimization parameters were the number of neurons, the weight cost, and learning rates. They evaluated their method on the turbofan engine degradation problem provided by NASA, i.e., the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) data sets, composed of multivariate temporal data coming from 21 sensors.

Another DL model recently investigated in RUL prediction is the CNN [25], a type of multi-layer feed-forward network originally conceived to recognize visual patterns from images [26], whose feature learning capabilities are enabled through the combination of multiple convolution and pooling layers. Babu et al. [27] reported the first attempt of predicting RUL using CNN-based feature learning combined with linear regression from multi-channel time series data. As the authors pointed out, the multiple layers composing the DL architecture effectively learned local salience representations of multi-channel signals (provided as two-dimensional input) also in different scales. They showed good RUL estimation performance on two publicly available data sets, the NASA C-MAPSS data set and the PHM 2008 Data Challenge data set.

Since sensor data involved in RUL estimation are traditionally arranged in a time-series form, either uni- or multi- variate, the use of Recurrent Neural Networks (RNNs) has been suggested to better deal with the sequential nature of these data [28]. However, as it is well-known, RNNs suffer from a vanishing/exploding gradient in presence of long-term time dependency. To overcome this drawback, Zheng et al. [29] suggested to estimate RUL with a Long Short-Term Memory (LSTM) network, which is a type of RNN able to effectively model sequential data, without suffering of long-term time dependency problems. To validate their LSTM model for RUL estimation, they used the C-MAPSS data Set, the PHM 2008 Challenge data set and the Milling data set.

Aiming to exploit the power of CNN to learn discriminative features from two-dimensional inputs, Li et al. [30] proposed a CNN deep learning architecture based on a time window approach able to handle multi-variate temporal information. To validate the effectiveness of their approach, they estimated the RUL of aero-engines using the NASA C-MAPSS data set. The achieved prediction performance was significantly better than the CNN approach presented by Babu et al. [27], and also comparable with the LSTM approach of Malhotra et al. [31] but employing simpler architecture and lower computing load.

The approaches reviewed so far receive as input time series of sensor data (e.g., vibration signals, engine data, physical properties, etc.), usually organized as 1D arrays, but often also as 2D arrays in the case of multi-channel data (i.e., multivariate time series). A special case of 2D representation is image data, traditionally used for visual inspection to assess component or system conditions. Recently, image data were used to automatically predict fault or degradation of machine components, with better results for larger enough data sets [32].

Since large data set are not always available within PHM applications, the problem of insufficient images for training CNN models has been addressed through transfer learning by Marei et al. [33], using microscope images of cutting tool flank. Essentially, transfer learning allows to reuse in a new domain, knowledge acquired from a similar or different domain. In practice, DL models pre-trained on large general-purpose image data sets (e.g., ImageNet [34], ILSVRC: ImageNet Large Scale Visual Recognition Challenge [35], CIFAR-10/CIFAR-100 [36], and so on) can be fine-tuned using available image data to perform prediction on new problems. However, the underlying assumption for transfer learning to work well is that the feature distributions across the two domains are the same.

DL models require an accurate setting of multiple DNN architectural parameters, which is a time-consuming and experience-intensive task. The parameter setting problem has been tackled by Mo et al. [37], adopting an evolutionary algorithm to find the optimal parameter configuration. Furthermore, they proposed a multi-head CNN structure followed by a LSTM network, pointing out the superiority of multi-head CNN models over

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

single-head multi-channel ones, since the former keep separate the extracted features whereas the latter mix them all together losing specialized features. They demonstrated the effectiveness of their solution using time series data from the NASA C-MAPSS data set.

In real-world settings, the collection of machinery health information might be challenging due to some kind of restrictions (e.g., small component size and narrow camera field-of-view, component partially hidden inside the machine, and so on), giving rise to partial or incomplete data. This problem, referred to as the partial observation problem, has been addressed by Li et al. [38] presenting a supervised attention mechanism for feature learning based on CNN and LSTM, followed by a regression layer for estimating the RUL of an industrial cutting wheel from images.

4. Materials and methods

This section details the materials and methods used in this study to predict the RUL of a punch tool from 3D scan data. More specifically, the next subsections deal with the following aspects: 1) the data acquisition system and experimental setup, 2) the 3D scanning process (i.e., pre-processing of 3D point clouds and feature extraction), 3) the adopted metrics to evaluate RUL prediction performance, 4) the DNN architectures generated by genetic optimization, 5) the genetic optimization technique, and 6) the Support Vector Regression (SVR) approach used as classic ML benchmark.

4.1. Experimental Setup and Data Acquisitions

The present study focuses on RUL prediction of punch tools mounted on a punching machine, like the one shown in Figure 1, used to process pump workpieces by making a punch on their upper end. The RUL prediction is based on 3D scan data acquired by a

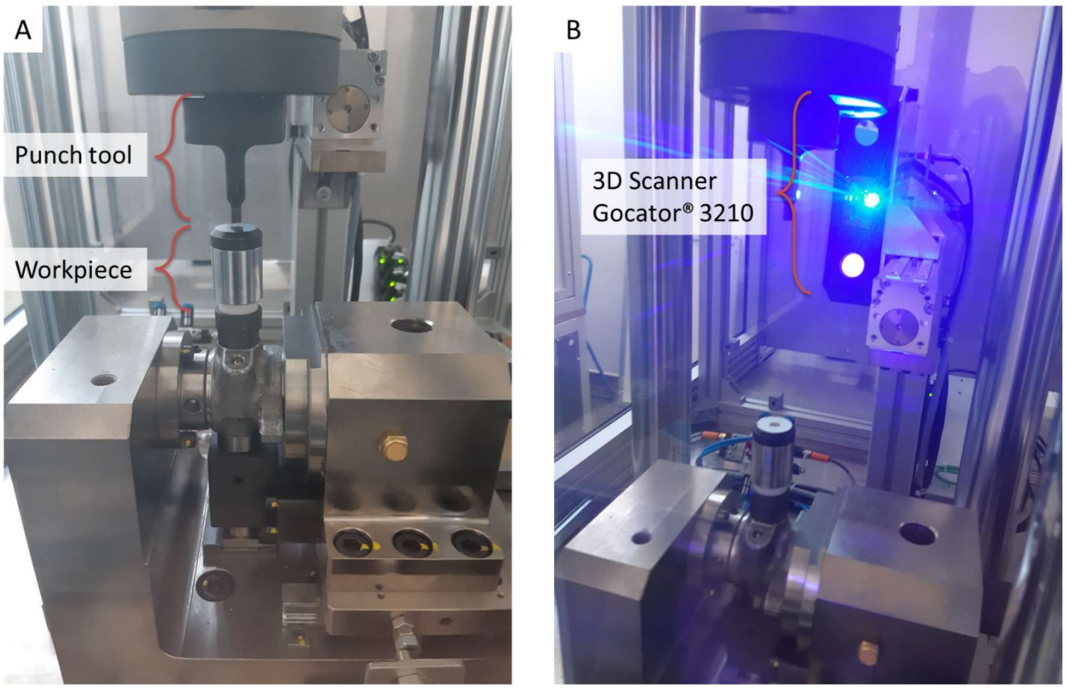

Figure 1. Punching machinery: A) punch tool and pump workpiece, B) Gocator® 3210 3D scanning sensor.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

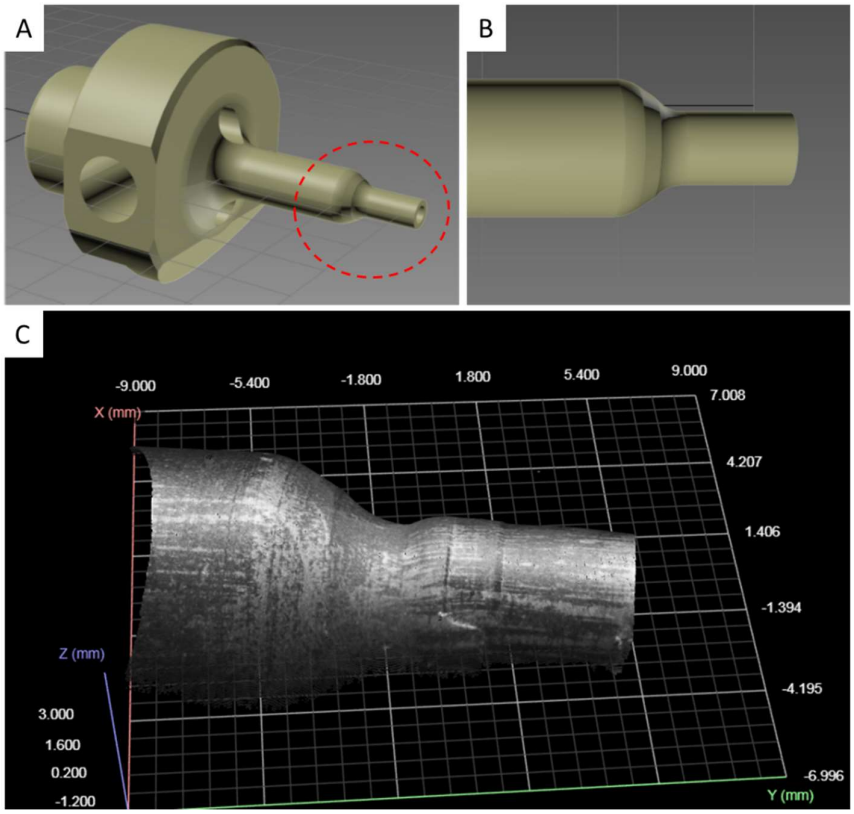

Figure 2. Punch tool 3D model. A) Full view of the punch tool with working region circled with a dashed red line. B) Detail of the working region. C) 3D point cloud with grayscale patches of the working region obtained from a 3D scan snapshot.

Gocator® 3210 [39] sensor, tightly clamped on the punching machine structure (Figure 1.B). The 3D scan sensor, equipped with a stereo camera of two megapixels and a blue LED projector, provides 3D point clouds in a single snapshot for accurate noncontact measurements down to 35 μm.

The punch tool consists of two cylindrical ends, of which the one having a larger diameter serves to clamp it to the machine, while the other is the working region. The 3D model of the punch tool is provided in Figure 2, in which the working region is highlighted by a dashed red line (Figure 2.A). During the machining cycles, the working surface of the punch (Figure 2.B) undergoes progressive deformations (Figure 2.C). 3D scans of these surface deformations, suitably processed using ML algorithms, can be exploited to predict the RUL of the punch tool.

During the experimentation phase, three identical punch tools were brought to the end of their life cycle, subjecting them to different loads. The first punch tool, P1, was operated with an incremental load ranging from a minimum of 10 kN to a maximum of 15 kN, obtaining a total of 714 scans performed every 50 pressing cycles. The second punch tool, P2, was tested with an incremental load ranging from 20 to 28 kN, for a total of 521 scans performed every 60 pressing cycles. The third punch, P3, was subjected to an incremental load ranging from 15 to 20 kN, generating a total of 779 scans captured every 50 pressing cycles. In such a way, a total amount of 2014 scans were produced in approximately six months.

4.2. 3D Scan Preprocessing

3D scans obtained as discussed above were preprocessed in the form of 3D point clouds. Due to reflections from metallic surface, raw 3D point clouds may be corrupted by artefacts, i.e., spurious 3D points. To overcome this issue, the first preprocessing step was to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

segment each raw point cloud into clusters, considering a minimum Euclidean distance between 3D points from different clusters, as represented in Figure 3. Then, the clusters were filtered based on the number of 3D points, keeping the two clusters with the greater number of points. The resulting 3D point cloud, provided in Figure 4, is composed of two main segments. The points located at y<10 form the working surface, whereas those located at y>30 represent the so-called best-fit surface used in the following step for point cloud registration.

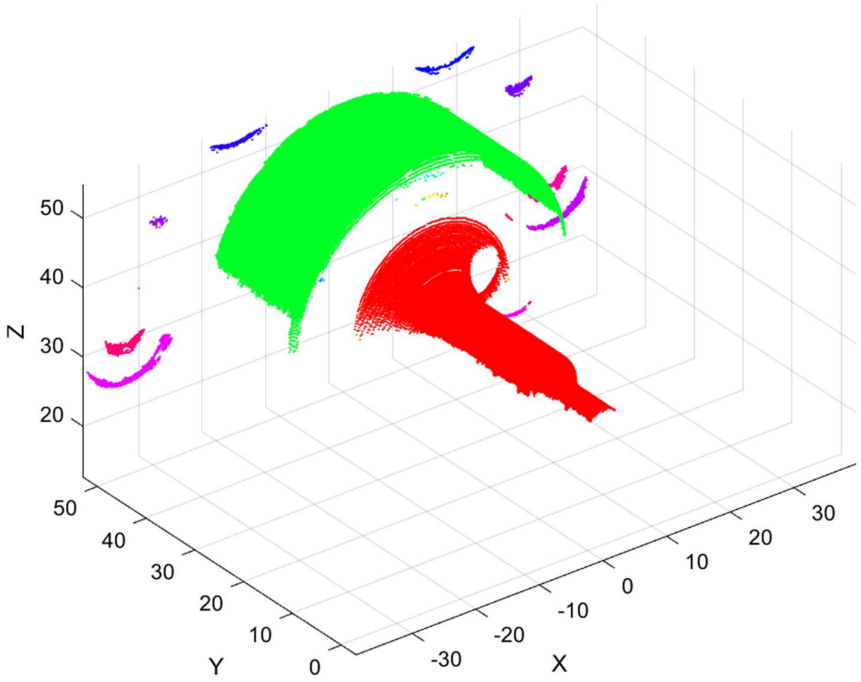

Figure 3. 3D point cloud segmented using the Euclidean distance between 3D points from different clusters. Segmented clusters are represented in different colors.

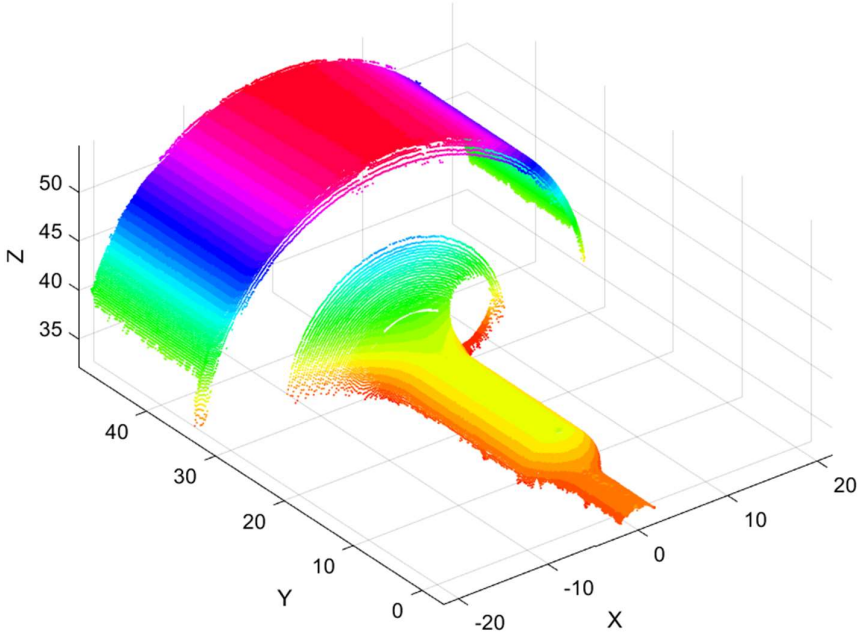

Figure 4. Filtered 3D point cloud after segmentation. Two main segments are visible.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

Although the 3D scan system is firmly clamped to the machine structure, continuous vibrations can produce slight misalignments between 3D point clouds. A rigid registration step was utilized to correct such misalignments. The semi-cylindrical surface with the largest diameter shown in Figure 4 (i.e., 3D points with y>30) was used as best-fit reference surface for registration, since this surface was the least subject to deformation during punching operations. The iterative-closest-point (ICP) algorithm, originally suggested by Besl and McKay [40], was used to register the best-fit surfaces of segmented point clouds.

Substantially, the ICP algorithm is an optimization process whose main goal is to find the locally best (in a least-square sense) rigid transformation by means of singular value decomposition (SVD) [41]. More specifically, the iterative process consists of the following main steps: 1) projection of the two point-clouds under registration, 2) estimation of the optimal rigid transformation via SVD, 3) application of the transformation to a subset of randomly selected points, 4) evaluation of the alignment via least median estimator [42], 5) if the alignment error is smaller than a prefixed threshold, the rigid transformation is applied to the whole point clouds, otherwise, the above steps are repeated.

Once registered, the point cloud was cropped to take only the working region, thus highlighting the surface deformation, as shown in Figure 5. In addition, in order to further highlight the working surface deformations, for each point belonging to the cropped point cloud the surface normal vector was considered [43], obtaining the normal representation shown in Figure 6.

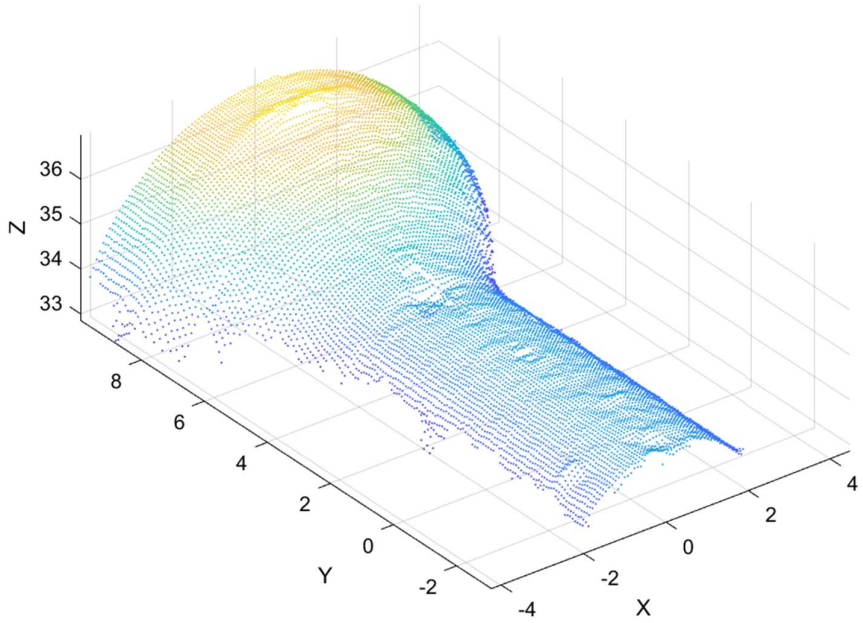

Figure 5. Working surface cropped from registered point cloud.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

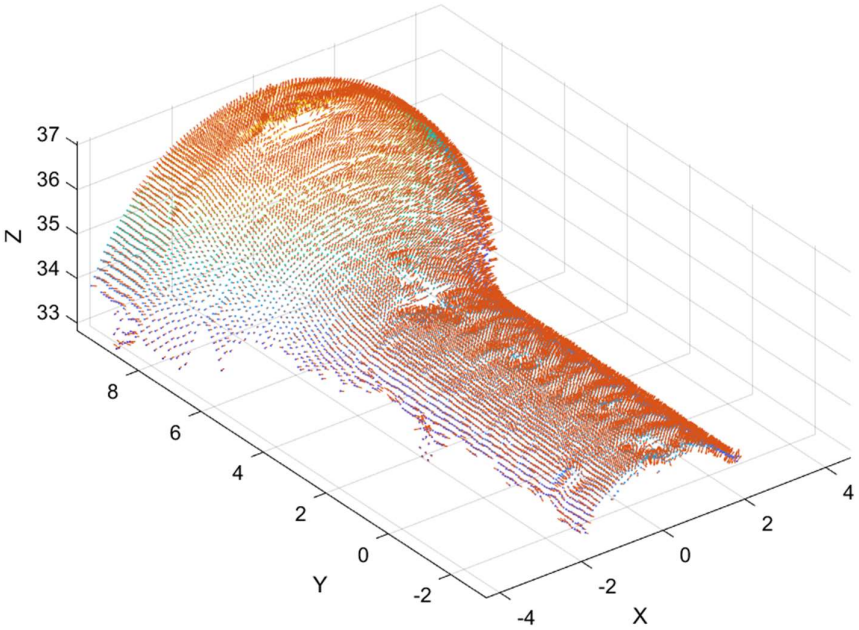

Figure 5. Surface normal vector representation of the working region obtained form the cropped point cloud provided in Figure 4.

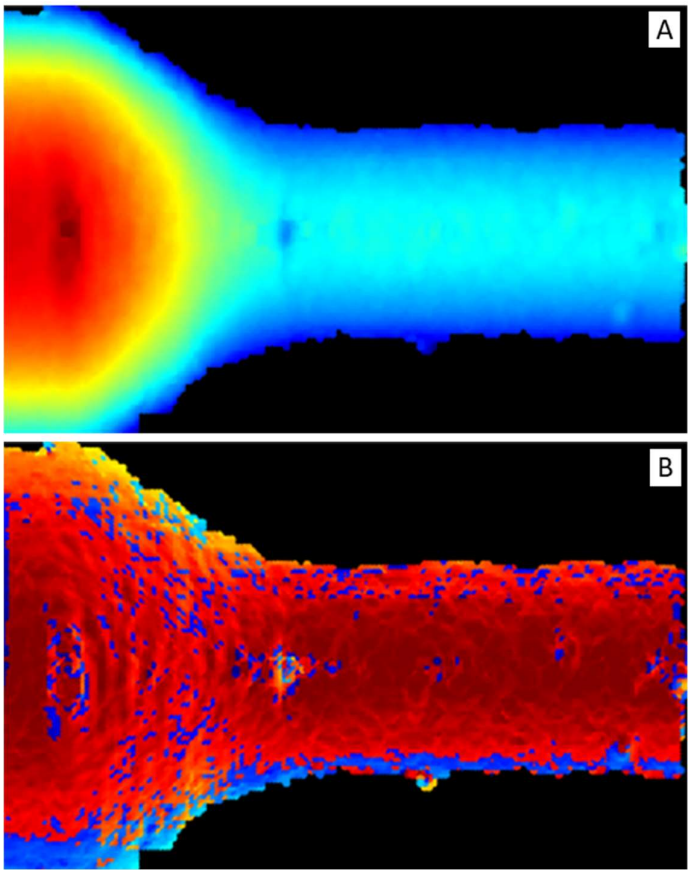

Figure 6. A) Depth map. Red color represents short distances and blue long ones. B) Normal map. Dark-red color represents vectors parallel to z-axis (i.e., pointing out of the image plane) and light-blue perpendicular ones.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

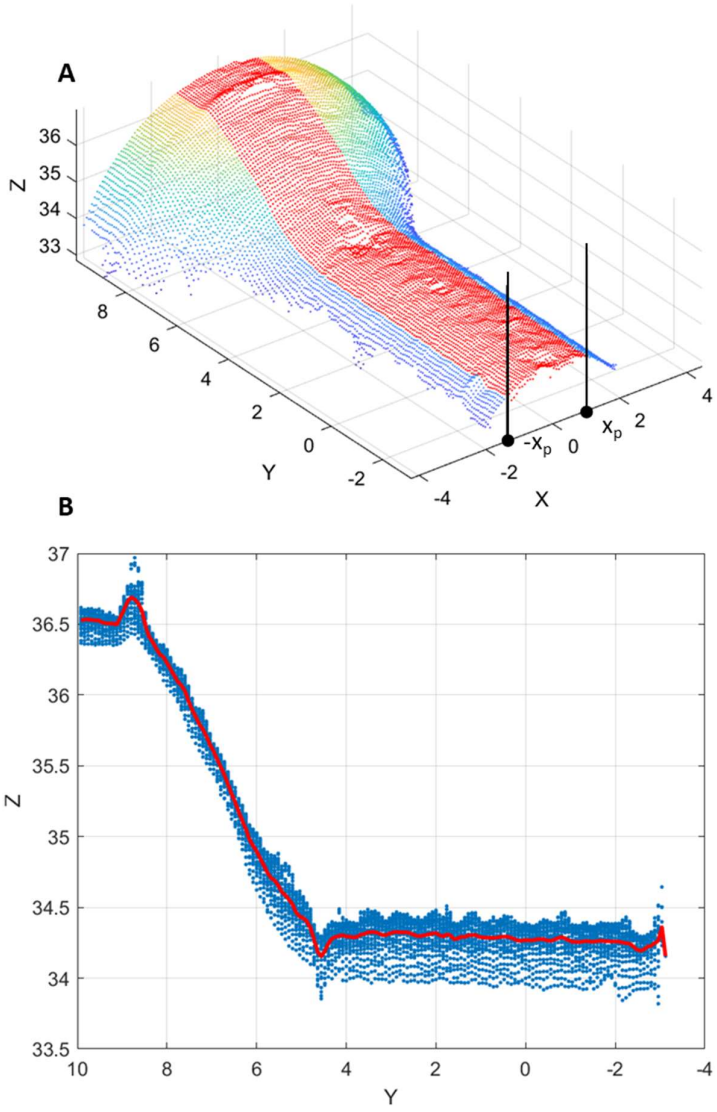

Figure 7. Longitudinal profile extraction. A) Longitudinal point-cloud region (red points) from which the longitudinal profile is estimated. B) Estimated longitudinal profile (red curve) from the point cloud projection on the YZ plane.

To exploit the most of DNN feature extraction capabilities, depth and normal vector representations were transformed into two-dimensional maps, reported in Figure 6, i.e., depth map (Figure 6.A) and normal vector map, or normal map (Figure 6.B), respectively.

The RUL prediction based on DFL was compared with that based on traditional ML methods (e.g., SVR). To do so, alongside the two-dimensional deformation representations mentioned above, one-dimensional representations were also considered, i.e., longitudinal profiles of the punch tool. The extraction process of longitudinal profiles, shown in Figure 7, consisted of three main steps. Firstly, a profile region was selected from the cropped point cloud by taking points $P = (x, y, z)$ such that $x \in [-x_p, x_p]$ (Figure 7.A). Secondly, the selected point-cloud region was projected onto the YZ plane. Thirdly, the longitudinal profile was estimated averaging the projected region along the Z-axis direction (Figure 7.B).

4.3. Evaluation Metrics

Three different evaluation metrics were adopted in this study, i.e., scoring function (SF), root mean square error (RMSE), and mean absolute percentage error (MAPE). The first two metrics were selected since they are commonly adopted in the literature on RUL

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

prediction [30], while the third was considered as it allows for more subtle evaluations than the other two.

Given a total number N of sampled machining cycles (i.e., 3D scans), let p_i be the RUL at the machining cycle i, p_i' the estimated version of p_i, and $E_i = p_i' - p_i$ the prediction error, the SF is defined as follows:

$$SF = \sum_{i=1}^{N} f_i,$$
$$\text{where } f_i = \begin{cases} e^{-\frac{E_i}{13}} - 1, & iif\ E_i < 0 \\ e^{\frac{E_i}{10}} - 1, & iif\ E_i \geq 0 \end{cases}. \quad (1)$$

Furthermore, the RMSE is defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} E_i^2}, \quad (2)$$

and, finally, MAPE is given by:

$$MAPE = \frac{1}{N}\sum_{i}^{N} \left|\frac{E_i}{p_i}\right|, \text{ for } p_i > 0. \quad (3)$$

4.4. DNN Architectures

The DNNs used in this study to capture the representation information from preprocessed inputs, i.e. depth and normal maps provided as color images, were based on CNN [44]. Both single- and double- head DNNs were investigated, whose general architectures

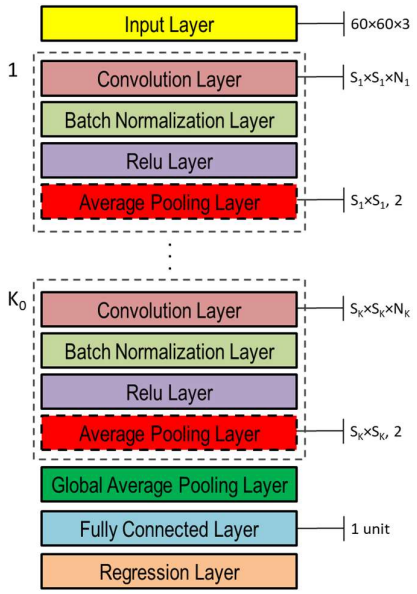

Figure 8. Single-head general architecture of CNN-based DNN for processing either depth or normal maps provided as color image inputs.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

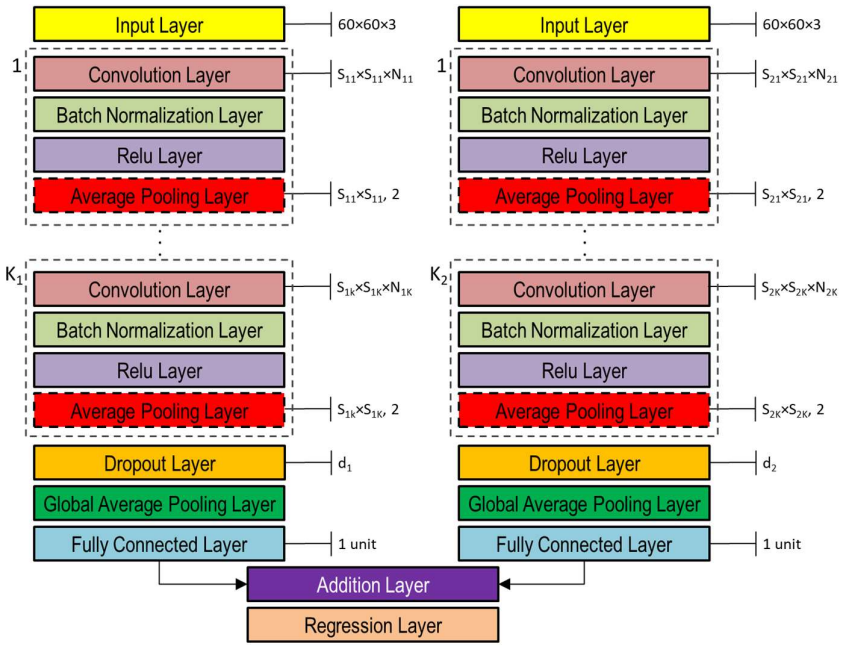

Figure 9. Double-head general architecture of CNN-based DNN for processing both depth and normal maps provided as color image inputs.

are shown in Figure 8 and Figure 9, respectively. Basically, they are composed by an input layer receiving depth or normal maps, resized to 60×60 pixels color (3-channels) images, followed by K_0 (K_1 and K_2 in the double-head case) blocks including the following four layers: 1) convolution, 2) batch normalization, 3) rectified linear unit (ReLU), and 4) global average pooling.

CNN feature learning is based on the convolution operation implemented by applying a kernel to input images (i.e., local receptive field) whose response provides the so-called feature map. Let $I = (I_1, I_2, I_3)$ be an input image, W a kernel of size $s \times s$ (a square kernel was considered in this study, but in general it may have a rectangular size) whose s^2 weights are adjusted in feedforward way, the feature map computed at $(x, y) \in I$ is given as follows:

$$F(x,y) = \sum_{i,j=1}^{s} I_h(x-i, y-j)W(i,j), \text{ with } h = 1,2,3; \tag{4}$$

where the summation is the convolution operation as the kernel Q slides over the image channel I_h. During the feedforward process, the output of a generic convolution layer at (x, y) is given as follows:

$$O_h(x,y) = \varphi\left(\sum_{i,j=1}^{s} I_h(x-i, y-j)W(i,j) + b\right), \text{ with } h = 1,2,3; \tag{5}$$

where $\varphi(\cdot)$ is the activation function used to introduce nonlinearity to feature maps, and b is term. In this study, the ReLU activation function was used which performs an element-wise threshold operation, i.e., sets to zero any input value less than zero:

$$\varphi(v) = \begin{cases} v, & iif\ v \geq 0 \\ 0, & iif\ v < 0 \end{cases}. \tag{6}$$

The batch normalization layer, inserted between convolution and nonlinear activation (ReLU) layers, allows to speed up training and regularize the network reducing initialization sensitivity, i.e., facilitates the convergence to good local minima without cumbersome initial parameter setting. Given an input element z_{ij} the batch normalization layer provides the following normalization:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

$$\hat{z}_{ij} = \frac{z_{ij} - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad (6)$$

where μ and σ are, respectively, mean and variance estimated over spatial, time and observation dimensions for each channel independently, whereas ϵ is a constant used to improve numerical stability if variance is too small.

The average pooling layer performs down-sampling by providing as output average values of its input, and thus reducing the connection number to the next layer, that helps to mitigate overfitting. The pool size adopted in this study was the same of the corresponding convolution layer in each CNN block, whereas the stride (i.e., step size with which the pooling layer scans through the input) was fixed at 2. The effect of the dropout layer is to turn off (set to zero), with probability p, a certain number of input elements randomly chosen. Such dropout operation has been shown to help prevent network overfitting [45]. In this study, the dropout layer was adopted only in the double-head case with $p = d_1$ and $p = d_2$ in the two heads, respectively (Figure 9).

The global average pooling layer provides further down-sampling by fully averaging the feature map. It is usually used before the final fully connected layer to reduce the size of activations, i.e., less weights, thus leading to a lower network size. The purpose of fully connected layer is to combine features to identify larger patterns. Thus, all neurons in fully connected layer are connected to all neurons of previous layer.

In case of classification problems, the last fully connected layer provides the features to perform classification, thus its output size is equal to the number of classes being classified. In case of regression problem, the output size is equal to the number of regression variables, which is one in this study. In both single- and double- head architectures, the continuous RUL value was finally estimated in the regression layer by minimizing the loss (i.e., not-normalized half mean squared error) that, in the case of this study with only one regression variable, reduces to E_i^2.

Generally, the learning process is casted as an optimization problem of minimizing the loss function. In this study, the stochastic gradient descent (SGD) [46] was used as optimization scheme. In SGD training, a mini-batch is stochastically selected at each time step, instead of using the entire training set, thus improving computing speed. Two relevant SGD parameters are initial learning rate η and momentum λ. If η is too low the training process takes a long time, whereas if it is too high the training might result suboptimal or divergent.

The momentum is a technique used in conjunction with SGD that adjusts the contribution of gradients at previous steps to determine the direction to proceed, instead of using only the gradient at the current step. If λ is equal to zero there is no contribution from previous steps, whereas if λ is one the contribution from previous steps is maximal. In this study, the network hyperparameters $(S_1, N_1, \dots, S_K, N_K, \eta, \lambda)$ in the single-head case, and $(S_{11}, N_{11}, \dots, S_{1K}, N_{1K}, d_1, S_{21}, N_{21}, \dots, S_{2K}, N_{2K}, d_2, \eta, \lambda)$ were determined by genetic optimization algorithm as discussed in the following subsection. Note that the average polling layer included in each CNN block is optional and its inclusion or exclusion was also considered among the network hyperparameters subject to optimization through binary array variables as detailed in the following subsection.

In addition to the architectures shown in Figure 8 and Figure 9, pre-trained networks were also evaluated. The adoption of pre-trained models offers multiple advantages, such as the possibility to exploit complex models without having to train them from scratch, even when little training data are available (used to fine-tune the pre-trained model) without running into overfitting problems, commonly found in presence of small training data sets. The technique underlying pre-trained models is called transfer learning [47] and, basically, allows to apply the knowledge already learned from one domain to another.

However, the transfer of knowledge from one domain to another is not always feasible. When the source domain is not sufficiently related to the destination domain, or when the transfer methodology is not able to take advantage of relationships between domains, this can lead to negative transfer [47]. For that reason, in this study, the most popular state-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

of-the-art pre-trained models reported in Table 1 were evaluated, and their prediction performance was compared with that of network architectures discussed above. Since such pre-trained networks are designed for classification problems, they were adapted by substituting the last three layers, i.e., global average pooling layer, softmax layer, classification layer, with a fully connected layer with one output neuron followed by a regression layer.

Table 1. Pre-trained networks evaluated for transfer learning.

Network	Input size	Parameters (10^6)
Squeezenet [48]	227 × 227 × 3	1.2
Googlenet [49]	224 × 224 × 3	7.0
Inceptionv3 [50]	299 × 299 × 3	23.9
Densenet201 [51]	224 × 224 × 3	20.0
Mobilenetv2 [52]	224 × 224 × 3	3.5
Resnet18 [53]	224 × 224 × 3	11.7
Resnet50 [53]	224 × 224 × 3	25.6
Resnet101 [53]	224 × 224 × 3	44.6
Xception [54]	299 × 299 × 3	22.9
Inceptionresnetv2 [55]	299 × 299 × 3	55.9
Shufflenet [56]	224 × 224 × 3	1.4
Nasnetmobile [57]	224 × 224 × 3	5.3
Nasnetlarge [57]	331 × 331 × 3	88.9
Darknet19 [58]	256 × 256 × 3	20.8
Darknet53 [58]	256 × 256 × 3	41.6
Efficientnetb0 [59]	224 × 224 × 3	5.3
Alexnet [60]	227 × 227 × 3	61.0
Vgg16 [61]	224 × 224 × 3	138.0
Vgg19 [61]	224 × 224 × 3	144.0

4.5. Genetic Optimization

The parameters of the network architectures shown in Figure 8 and Figure 9 were optimized by means of genetic optimization technique. GAs are population-based optimization methodologies that take their cue from the evolutionary process of living beings, i.e., the metaphor of natural biological evolution [62]. GAs iteratively implement a series of operations to manipulate populations of candidate solutions (i.e., chromosomes) to produce new solutions by means of genetic functionals such as reproduction, crossover, and mutation. Ultimately, they are inspired by Darwin's theory of evolution and relative principles of reproduction, genetic recombination, and survival of the fittest. The population of chromosomes (i.e., candidate solutions) is evaluated through the attribution of a score carried out through a so-called fitness function, the formulation of which depends on the specific optimization problem.

In this study, the fitness function f built network architectures and evaluated them, providing as output the MAPE value obtained from testing. Thus, the optimization problem was formulated as follows:

$$\begin{gathered} \text{minimize } f(\boldsymbol{z}), \\ \text{with } \boldsymbol{z} = (z_1, z_2, \dots, z_M) \text{ such that } z_i^L \le z_i \le z_i^U, i = 1, \dots, M, \end{gathered} \tag{7}$$

were $(z_1, z_2, \dots, z_M)$ are optimization variables, continuous or integer valued, bounded between $(z_1^L, z_2^L, \dots, z_M^L)$ and $(z_1^U, z_2^U, \dots, z_M^U)$, respectively, defining candidate network architectures as better explained in the following. The number of optimization variables was $M = 5$ for single-head architectures (Figure 8), and $M = 10$ for double-head architectures (Figure 9). For both architectures, the number of CNN blocks ranged from four to six, i.e., $K_i \in \{4,5,6\}, i = 1,2,3$.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

In the single-head case, the optimization variables were $\boldsymbol{z} = (x_1, x_2, x, x_4, x_5)$, where $x_1 \in \mathbb{N}$ represented the filter sizes, $x_2 \in \mathbb{N}$ specified the number of filters, $x_3 \in \mathbb{N}$ the presence or not of average pooling layers, $x_4 \in \mathbb{R}$ the initial learning rate, and $x_5 \in \mathbb{R}$ the momentum. In the double-head case, instead, the optimization variables were $\boldsymbol{z} = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10})$, and $y_1, y_2, y_3 \in \mathbb{N}$ represented the filter sizes, number of filters, and presence or not of average pooling layers for the first head, whereas $y_4, y_5, y_6 \in \mathbb{N}$ were the filter sizes, number of filters, and presence or not of average pooling layers for the second head. $y_7 \in \mathbb{R}$ was the initial learning rate, $y_8 \in \mathbb{R}$ the momentum, $y_9 \in \mathbb{R}$ the dropout probability for the first head, and $y_{10} \in \mathbb{R}$ the dropout probability for the second head.

Regarding the convolutional filter sizes, i.e., x_1 (single-head) or y_1 and y_4 (double-head) variables, odd square dimensions ranging from 3×3 to 29×29 were evaluated by considering all possible combinations taken K_i $(i = 0,1,2)$ at a time. For example, in the case of $K_0 = 4$, $x_1 = 1$ corresponded to $(S_1, S_2, S_3, S_4) = (3,5,7,9)$, $x_1 = 2$ to $(S_1, S_2, S_3, S_4) = (3,5,7,11)$, $x_1 = 3$ to $(S_1, S_2, S_3, S_4) = (3,5,7,13)$, etc., x_{1001} to (23,25,27,29), after that it continued in reverse order, i.e., x_{1002} corresponded to (29,27,25,23), x_{1003} corresponded to (29,27,25,21), and so on. Regarding the number of filters, i.e., x_2 (single-head) or y_2 and y_5 (double-head) variables, power of two between 8 and 256 were considered in incremental order. Thus, for example, in the case $K_0 = 5$, $x_2 = 1$ gave $(N_1, N_2, N_3, N_4, N_5) = (8,8,8,8,8)$, $x_2 = 2$ gave $(N_1, N_2, N_3, N_4, N_5) = (8,8,8,8,16)$, $x_2 = 3$ gave $(N_1, N_2, N_3, N_4, N_5) = (8,8,8,8,32)$, and so on.

Since the average pooling layers, depicted in Figure 8 and Figure 9 with dashed lines, were optional, their presence or absence were regulated by variables x_3 (single-head) or y_3 and y_6 (double-head), representing the configurations of a binary array $\{0,1\}^{M_i}$, where 1 in j-th position (with $j = 1, \dots, M_i$) indicated the presence of average pooling layer at the end of the j-th CNN block. For example, in the case of $K_0 = 6$, $x_3 = 1$ corresponded to $(0,0,0,0,0,0)$, $x_3 = 2$ corresponded to $(0,0,0,0,0,1)$, $x_3 = 3$ to $(0,0,0,0,1,1)$, and so on. All previously discussed optimization variables are summarized in Table 2.

Table 2. Lower and upper bounds of all optimization variables.

Variable	M_i	Lower bound	Upper bound
Filter size (x_1, y_1, y_4)	4	1	2002
"	5	1	4004
"	6	1	6006
Number of filters (x_2, y_2, y_5)	4	1	35
"	5	1	126
"	6	1	462
Pooling positions (x_3, y_3, y_6)	4	1	16
"	5	1	32
"	6	1	64
Initial learning rate (x_4, y_7)	4, 5, 6	10^{-4}	0.1
Momentum (x_5, y_8)	4, 5, 6	0	1
Dropout probability (y_9, y_{10})	4, 5, 6	0	1

In this study, both suggested DNNs (Figure 8 and Figure 9) and pretrained ones (Table 1) were implemented and evaluated using the MatWorks® Deep Learning Toolbox (v 14.2, R2021a) [63]; whereas, genetic optimization was performed using the MatWorks® Optimization Toolbox (v 9.1, R2021a) [64].

4.6. SVR based estimation

As a further comparison, the previously presented DNN-based models were compared with more traditional ML methods such as SVR. Since the presence of irrelevant or redundant information could slow down or make prediction algorithms less accurate, it

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

is necessary, before learning model, to distinguish between relevant and unnecessary features. For this reason, the first step was to reduce the dimensionality of the profile data (Figure 7.B) using the PCA approach [65].

The profile data were represented in YZ plane by curves $\Psi_k = \{(y_i^k, z_i^k) \in \mathbb{R}^2, i = 1, \dots, N_{P_k}\}$, $k = 1, \dots, N$, with N_{P_k} typically ranging between 156 to 164 depending on the specific point-cloud considered. After the PCA application, the reduced profile data were given by $\bar{\Psi}_k = \{(\bar{y}_i^k, \bar{z}_i^k) \in \mathbb{R}^2, i = 1,2\}$, since the percentage of variance explained by the first two principal components was of 100%. Ultimately, profile feature data used to train and test the SVR model was written as

$$\boldsymbol{P} = \begin{bmatrix} \bar{y}_1^1 & \bar{z}_1^1 & \bar{y}_2^1 & \bar{z}_2^1 \\ & \vdots & & \\ \bar{y}_1^N & \bar{z}_1^N & \bar{y}_2^N & \bar{z}_2^N \end{bmatrix} \in \mathbb{R}^{N,4}. \tag{8}$$

To achieve a good compromise between processing speed and accuracy, in this study, the epsilon-insensitive SVR (i.e., ε −SVR) [66, 67] was adopted, in which the epsilon parameter controls the amount of error allowed to the model. Given training data $\boldsymbol{P}_i = (\bar{y}_1^i, \bar{z}_1^i, \bar{y}_2^i, \bar{z}_2^i)^{\boldsymbol{T}} \in \mathbb{R}^{\boldsymbol{4}}$, the goal is to find a function $g(\boldsymbol{P}_i)$ that deviates from RUL values $p_i \in \mathbb{R}$ by an amount no greater than ϵ while at the same time being as flat as possible.

In the linear case, assuming that the training data set is composed of $N_T < N$ profiles $\boldsymbol{P}_i$ and corresponding RUL values p_i with $i = 1, \dots, N_T$, the linear function takes the form $g(\boldsymbol{P}_i) = \langle \boldsymbol{\pi} \cdot \boldsymbol{P}_i \rangle + b$, where $\langle \cdot \rangle$ is the dot product, and $\boldsymbol{\pi} \in \mathbb{R}^{\boldsymbol{4}}$ such that $\langle \boldsymbol{\pi} \cdot \boldsymbol{\pi} \rangle$ is minimum to ensure flatness. The problem can be stated in term of convex optimization as follows:

$$\begin{gathered} \text{minimize } \|\boldsymbol{\pi}\|^2, \text{ subject to} \\ \forall i = 1, \dots, N_T : |p_i - (\langle \boldsymbol{\pi} \cdot \boldsymbol{P}_i \rangle + b)| \leq \epsilon. \end{gathered} \tag{9}$$

Since a function f satisfying these constraints for all points may not exist, in practice slake variables (ξ_i, ξ_i^*) are introduced, analogously to the concept of "soft margin" in SVM. With the addition of the slake variables the problem (9) becomes [66]:

$$\begin{gathered} \text{minimize } \|\boldsymbol{\pi}\|^2 + C \sum_{i=1}^{N_T} (\xi_i + \xi_i^*), \text{ subject to} \\ \forall i = 1, \dots, N_T : \begin{cases} p_i - \langle \boldsymbol{\pi} \cdot \boldsymbol{P}_i \rangle - b \leq \epsilon + \xi_i \\ \langle \boldsymbol{\pi} \cdot \boldsymbol{P}_i \rangle + b - p_i \leq \epsilon + \xi_i^* \\ \xi_i \geq 0 \\ \xi_i^* \geq 0 \end{cases}, \end{gathered} \tag{10}$$

where the positive parameter C controls the penalty imposed on observations that fall outside the ϵ margin, playing a regularizing role to prevent overfiting.

In the nonlinear case, the dot product is replaced with a kernel function $G(\cdot,\cdot)$ which maps training data to a high-dimensional space. Some popular kernel functions evaluated in this study are reported in Table 3.

Table 3. Kernel functions evaluated in this study.

Kernel	$\boldsymbol{G(x, y)}$
Linear	$\langle x \cdot y \rangle$
Gaussian	$e^{-\|x-y\|^2}$
Polynomial	$(1 + \langle x \cdot y \rangle)^q$ with $q \in \mathbb{N}\backslash\{0,1\}$

5. Results

The performance results of the genetically optimized network architectures are provided in Table 4. In the single-head case, the convention used for the model name is a prefix "go" which stands for genetically optimized, followed by the number of CNN

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

blocks and the suffix "normal" or "depth" depending on the type of map the model was tested on. Thus, for instance, the name of the model genetically optimized with four CNN blocks and tested on normal maps is "go4normal". Instead, in the double-head architectures, since they were tested on both normal and depth maps, the naming convention consists of the suffix "go" followed by the number of blocks for the two heads, for example "go4+4" indicates a model with four blocks for each head. In addition to the three metrics defined in (1), (2) and (3), the last column of Table 4 provides the time required to train each model.

The population size was 50 in single-head architectures (5 optimization variables) and 200 in double-head ones (10 optimization variables), for a total number of 10,000 and 40,000 iterations, respectively. For each candidate architecture, the model was trained for 100 epochs, randomizing the validation data set to each epoch. To achieve the final performance, the results of the top 100 architectures based on the fitness function (i.e., the MEPA metric) were averaged.

The pre-trained models (Table 1) tested on normal and depth maps are provided in Table 5 and Table 6, respectively. The last columns of these tables report the fine-tuning time (FTT), i.e., the time elapsed to fine-tune each pre-trained model for 30 epochs. Also in this case, performance results were averaged by repeating training and testing 100 times for each model.

As regards the classical approaches based on SVR, three kernels, i.e., linear, Gaussian and polynomial of order 3, 4, 5 and 6, were tested. The performance results obtained with the approaches based on SVR are reported in Table 7. In this study , SVR was considered as a benchmark for evaluating the goodness of DNN-based models.

A comprehensive overview of all achieved results is shown in Figure 10. As can be seen from this figure, the models that performed best are the genetically optimized ones (numbers from 1 to 15), while most of the pre-trained models (from no. 16 to 51) performed worse than the SVR algorithms (from no. 52 to 57).), with the exception of the pre-trained models: googlenet (no. 17), vgg16 (no. 32), vgg19 (no. 33) on maps of normal vectors, and the pre-trained models: googlenet (no. 35), alexnet (no. 49), and vgg19 (no. 51) on depth maps.

Figure 10 also reports training times (TTs) of genetically optimized and SVR-based models and FTTs of pre-trained models, revealing on average longer times for the pre-trained models (from no. 18 to 51), on average shorter times for the genetically optimized models (from no. 1 to 15), and very short times for the SVR models (from no. 52 to 57).

Finally, the network configurations of genetically optimized single- and double- head DNN architectures are reported in Table 8 and Table 9, respectively. The reported parameters refer to best-fit models resulting from the genetic optimization process. The last columns show the number of learned parameters (learnables) of each architecture. The genetic optimization process lasted an average of 53 hours for each single-head architecture and approximately 632 hours for each dual-head architecture. The total duration of the genetic optimization process was approximately 250 days on a computer system equipped with Graphics Processing Unit (GPU) and configured as follows: Intel® Core™ i7-5820K CPU @ 3.30GHz, 16 GB DRAM, and NVIDIA GeForce GTX TITAN X GPU (Maxwell family) with 12 GB GRAM.

Table 4. RUL prediction performance obtained with genetically optimized networks tested on both depth and normal maps.

No.	Model name	MAPE	RMSE	SF	TT (sec)
1	go4normal	0.063	0.036	0.301	63.793
2	go5normal	0.065	0.037	0.313	18.351
3	go6normal	0.083	0.035	0.282	16.928
4	go4depth	0.102	0.063	0.513	13.647
5	go5depth	0.083	0.049	0.393	18.603
6	go6depth	0.058	0.036	0.312	13.410

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

7	go4+4	0.097	0.039	0.349	31.233
8	go4+5	0.141	0.057	0.493	125.250
9	go4+6	0.154	0.064	0.569	33.631
10	go5+4	0.123	0.048	0.427	96.948
11	go5+5	0.162	0.062	0.545	58.817
12	go5+6	0.129	0.053	0.469	58.204
13	go6+4	0.158	0.067	0.593	37.651
14	go6+5	0.237	0.082	0.762	196.594
15	go6+6	0.251	0.085	0.812	35.540

Table 5. RUL prediction performance obtained with pre-trained networks tested on normal maps.

No.	Model name	MAPE	RMSE	SF	FTT (sec)
16	squeezenet	2.080	0.363	4.044	32.875
17	googlenet	0.452	0.112	1.077	18.947
18	inceptionv3	3.402	0.367	3.839	121.734
19	densenet201	1.802	0.559	5.453	758.300
20	mobilenetv2	2.692	0.608	6.471	94.666
21	resnet18	3.053	0.638	6.719	48.073
22	resnet50	1.854	0.426	4.086	140.182
23	resnet101	1.193	0.432	4.095	270.903
24	xception	1.825	0.365	1.550	833.500
25	inceptionresnetv2	2.691	0.555	5.365	755.150
26	shufflenet	2.673	0.905	9.534	73.828
27	nasnetmobile	1.597	0.412	4.276	352.860
28	darknet19	1.623	0.340	1.332	108.202
29	darknet53	1.384	0.485	1.870	565.150
30	efficientnetb0	1.408	0.338	3.306	195.668
31	alexnet	1.048	0.160	1.482	37.516
32	vgg16	0.615	0.311	1.177	310.489
33	vgg19	0.698	0.484	1.821	405.430

Table 6. RUL prediction performance obtained with pre-trained networks tested on depth maps.

No.	Model name	MAPE	RMSE	SF	FTT (sec)
34	squeezenet	3.789	0.321	3.175	36.501
35	googlenet	0.416	0.114	1.142	56.326
36	inceptionv3	2.200	0.312	3.067	267.350
37	densenet201	1.657	0.527	5.387	785.250
38	mobilenetv2	2.108	0.452	4.768	104.493
39	resnet18	1.602	0.448	4.576	48.366
40	resnet50	1.181	0.312	2.963	147.724
41	resnet101	1.838	0.633	7.828	283.090
42	xception	1.790	0.409	1.725	915.500
43	inceptionresnetv2	1.530	0.266	2.519	745.000
44	shufflenet	2.062	0.694	7.401	74.328
45	nasnetmobile	1.309	0.324	3.329	345.805
46	darknet19	9.471	3.291	14.630	126.012
47	darknet53	0.878	0.423	1.557	580.700
48	efficientnetb0	0.870	0.274	2.761	183.381
49	alexnet	0.822	0.311	2.921	35.722
50	vgg16	1.095	0.616	2.476	312.283
51	vgg19	0.648	0.452	1.683	409.718

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 August 2021 doi:10.20944/preprints202108.0272.v1

Table 7. RUL prediction performance obtained with SVR algorithms tested on surface profiles.

No.	Model name	MAPE	RMSE	SF	TT (sec)
52	linear	1.073	0.135	1.444	4.667
53	gaussian	1.190	0.182	1.673	4.117
54	polynomial3	1.107	0.165	1.502	5.053
55	polynomial4	0.909	0.124	1.180	5.726
56	polynomial5	0.862	0.113	1.134	5.734
57	polynomial6	0.857	0.120	1.179	5.807

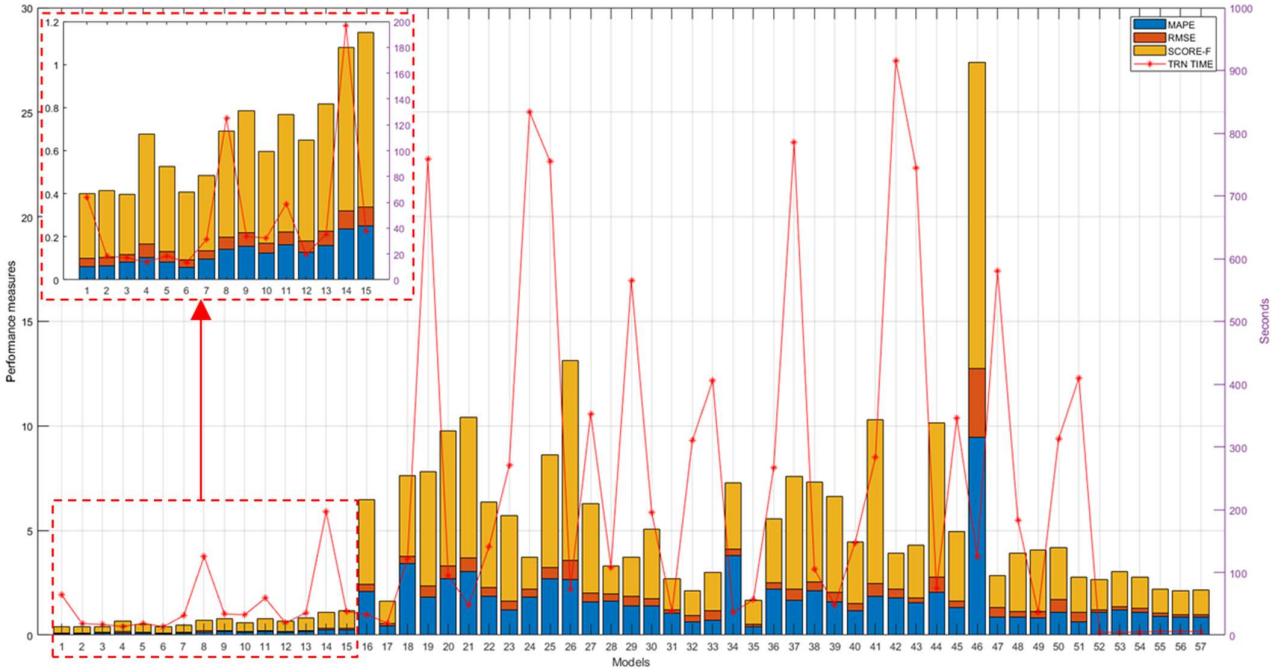

Figure 10. Performance measures (left y-axis) and TTs/FTTs (right y-axis) of all evaluated models.

Table 8. Network configuration of genetically optimized single-head DNN architectures.

No.	Model name	Network architecture	Parameters (10^6)
1	go4normal	**S** = (29,25,19,17) **N** = (32,32,32,32) **P** = (0,0,0,0) ILR = 0.0059 M = 0.8634	1.39
2	go5normal	**S** = (5,7,19,23,29) **N** = (32,32,32,64,64) **P** = (1,0,0,0,0) ILR = 0.0068 M = 0.8049	4.95
3	go6normal	**S** = (29,25,23,19,11,7) **N** = (16,32,32,32,64,64) **P** = (1,1,1,1,0,0) ILR = 0.0047	1.72

 doi:10.20944/preprints202108.0272.v1

		M = 0.7572	
4	go4depth	**S** = (29,23,21,15) **N** = (16,16,16,16) **P** = (1,1,1,1) ILR = 0.0074 M = 0.6636	0.347
5	go5depth	**S** = (29,27,15,11,3) **N** = (32,32,32,32,32) **P** = (1,1,1,0,0) ILR = 0.0054 M = 0.6893	1.19
6	go6depth	**S** = (3,5,15,21,23,27) **N** = (32,64,64,64,64,64) **P** = (1,0,0,0,0,0) ILR = 0.0077 M = 0.7702	7.93

Table 9. Network configuration of genetically optimized double-head DNN architectures.

No.	Model name	Network architecture	Parameters (10^6)
7	go4+4	**S1** = (5,15,21,29), **S2**=(29,27,25,17) **N1** = (16,16,16,32), **N2** = (16,16,16,16) **P1** = (1,1,1,1), **P2** = (1,1,1,1) d1 = 0.6791, d2 = 0.2646 ILR = 0.0086 M = 0.6499	1.06
8	go4+5	**S1** = (27,23,19,3), **S2**=(29,25,23,13,7) **N1** = (32,32,32,32), **N2** = (16,64,64,64,64) **P1** = (1,0,0,0), **P2** = (0,0,0,0,0) d1 = 0.2452, d2 = 0.2494 ILR = 0.0084 M = 0.5046	4.73
9	go4+6	**S1** = (5,23,27,29), **S2**=(27,25,23,17,13,9) **N1** = (16,32,32,32), **N2** = (16,16,16,32,64,64) **P1** = (1,0,0,0), **P2** = (0,0,0,0,0,0) d1 = 0.2659, d2 = 0.4718 ILR = 0.0069 M = 0.6599	7.76
10	go5+4	**S1** = (27,25,23,21,17), **S2** = (29,25,11,5) **N1** = (16,32,32,32,64), **N2** = (32,32,32,32) **P1** = (0,0,0,0,0), **P2** = (0,0,0,0) d1 = 0.598, d1 = 0.282 ILR = 0.008 M = 0.804	2.81
11	go5+5	**S1** = (25,21,19,15,7), **S2** = (29,21,15,13,3) **N1** = (64,64,64,64,64), **N2** = (16,32,32,64,64) **P1** = (1,1,1,0,0), **P2** = (0,0,0,0,0) d1 = 0.1296, d2 = 0.1075 ILR = 0.0089 M = 0.4570	5.41
12	go5+6	**S1** = (9,11,13,25,27), **S2** = (7,11,17,19,21,27) **N1** = (16,16,32,64,64), **N2** = (16,32,64,64,64,64) **P1** = (1,0,0,0,0), **P2** = (0,0,0,0,0,0) d1 = 0.425, d1 = 0.355	11.32

 doi:10.20944/preprints202108.0272.v1

		ILR = 0.006 M=0.464	
13	go6+4	**S1** = (5,15,19,21,23,29), **S2**=(11,17,19,25) **N1** = (16,16,32,32,32,32), **N2** = (16,16,32,32) **P1** = (1,0,0,0,0,0), **P2** = (1,1,0,0) d1 = 0.7385, d2 = 0.5250 ILR = 0.0039 M = 0.1515	3.00
14	go6+5	**S1** = (29,27,19,17,13,5), **S2**=(7,17,21,27,29) **N1**=(64,64,64,64,64,64), **N2**=(16,32,32,32,32) **P1** = (0,0,0,0,0,0), **P2** = (1,0,0,0,0) d1 = 0.8581, d2 = 0.5764 ILR = 0.0074 M = 0.5507	8.82
15	go6+6	**S1**=(3,13,15,21,25,29), **S2**=(3,11,13,17,21,25) **N1**=(16,16,16,32,64,64),**N2**=(16,16,64,64,64,64) **P1** = (1,0,0,0,0,0), **P2** = (1,0,0,0,0,0) d1 = 0.7867, d2 = 0.6495 ILR = 0.0046 M = 0.6290	10.81

5. Discussion

The use of profilometric scanning sensors allows to appreciate surface deformations with micrometric precision in the form of 3D point clouds. On the other hand, the organization of 3D point clouds into bidimensional image-like maps, as proposed in this study, enables to make the most of the potential of CNN-based DNN architectures, originally designed to process image data. Furthermore, in the case of RUL depending on surface deformations, two-dimensional maps offer the advantage of representing the state of system (i.e., punch tool in this study) deterioration in a cumulative way. Therefore, under such conditions, the RUL can be reliably estimated from a single image, that is from a single depth or normal map.

Pre-trained networks with transfer learning are advantageous since they allow to deal with small training data sets and to overcome the often-cumbersome process of generate problem-specific networks. However, they are not always suitable, especially when data distributions are very dissimilar between source and target domains. The definition of the most suitable DNN architecture for the problem under consideration, however, is not an easy task. In general, it involves identifying various configuration parameters (hyperparameters) through a trial-and-error process. The transfer learning method offers the indisputable advantage of simplifying this often-cumbersome process of generating *ad-hoc* DNN architectures. In addition, pre-trained models require the use of a small amount of training data for fine-tuning, allowing to address the additional problem of reduced amount of training data [33].

Keeping in mind the aforementioned advantages, in this study, the transfer learning technique was evaluated in correspondence with the main pre-trained models, as reported in Table 1, with both depth and normal maps. However, the performance results obtained were generally lower than the more traditional SVR approach taken as a reference (Figure 10). Only the pre-trained models googlenet (MAPE equal to 0.452 with normal maps and 0.416 with depth maps), vgg16 (MAPE = 0.615 with normal maps), vgg19 (MAPE equal to 0.698 with normal maps and 0.648 with depth maps) and alexnet (MAPE = 0.822 with depth maps) performed slightly better than the SVR approach (MAPE = 0.857 with polynomial kernel of order 6), but requiring on average 35 times more time for fine-tuning than SVR requires for training.

 doi:10.20944/preprints202108.0272.v1

In this study, the problem of defining *ad-hoc* (problem-specific) architectures was addressed by resorting to genetic optimization. In this way, a total amount of 15 architectures were optimized, of which 6 were single-headed (three for each type of map), shown in Table 8, and 9 double-headed, as shown in Table 9. The performance results reported in Figure 10 (see the magnification in the upper left corner) confirm the superiority of the genetically optimized architectures over the pre-trained ones. The drop in performance found with transfer learning method can be explained by the fact that feature distributions across the two domains, source and target, were very different from each other. The pre-trained models (Table 1), in fact, were pre-trained using mostly "natural" images, while the images proposed in this study were obtained by mapping 3D point clouds in order to represent depths and normal vectors to the punch tool surface, resulting in "artificial" images with false colors.

Among the genetically optimized architectures, the single-headed models performed better than double-headed ones, with a slight predominance of models trained and tested on depth maps over those evaluated with normal maps. These results indicate that the use of two-headed models is not beneficial, and it is probably explained by poor correlations between features extracted from the two different types of maps, depth and normal, in representing deformation-induced degradation.

The results achieved in this study are in line with the state of the art in the literature. In particular, regarding CNN-based studies with image datasets, one of the best results presented in the literature was reported by Wu et al. [32]. In their study, the authors reported an average MAPE of 0.0476 which, however, was obtained with a very large data set consisting of 8400 images, while the TT was between 1.8 and 32.6 hours (the authors have not provided specifications of used computing system). On the other hand, in the case of small data sets, one of the best results reported in the literature is that of Marei et al. [33] who achieved an average RMSE of 0.1654 with the Resnet18 pre-trained model on a data set of 327 images, requiring 3358.4 seconds for fine-tuning on an NVIDIA GPU with 8GB Ram. However, it should be kept in mind that images of deteriorating components or parts of them (i.e., real world images, often obtained under a microscope and by stopping the machining system) were used in those studies. Therefore, the adaptation (or transfer learning) of pre-existing (or pre-trained) CNN models was feasible, considering the similarity of feature distributions between domains. In the case, instead, of data sets consisting of time-series sensor data, Mo et al. [37] reported an average RMSE of 11.28 with the NASA C-MAPSS data set.

The proposed system has been conceived to be versatile, working in a completely automatic way. There is no need to disassemble the punch tool or stop the punch machine to capture scans. The used 3D sensor, attached to the punching machine, scans at regular intervals of pressing stops. Furthermore, it is important to note that both depth, normal maps and longitudinal profiles allow to estimate the punch tool RUL in a single-shot, i.e., a single profile or map accounts for all deformations occurred up to that moment. This allows to avoid processing long sequences of profiles or maps, reducing computational load and network architecture complexity.

Although the optimization process takes a long time, it only needs to be performed once for the type of punch tool used. In this study, three different punch tools were tested, characterized by different deformation modes, using the same network architectures. An aspect that deserves further investigation concerns the verification of whether the proposed architectures are also valid for predicting RUL of systems other than the studied punch tool, but whose degradation still depends on the work surface deformation.

6. Conclusions

In this study, a DNN-based RUL prediction framework for punch tool, whose deterioration is due to surface deformation, was investigated. The main results achieved are threefold as indicated below. Firstly, the surface deformation of the punch tool was represented through the definition of depth and normal vector maps, obtained from point

clouds of 3D scans. Secondly, the RUL prediction was estimated considering the main pre-trained models, obtaining lower or slightly higher performance than SVR-based classic ML, due to different distribution of features between the transfer learning domains. Thirdly, genetically optimized architectures based on variable number of CNN blocks, both single- and double-headed, were generated, achieving superior performance to pre-trained models and in line with the state-of-the-art in the literature.

Ongoing and future research focuses on experimentation of depth and normal vector maps in combination with genetically optimized DNN architectures for the RUL prediction of other systems whose deterioration depends on surface deformations.

Author Contributions: Conceptualization, G.D. and A.L.; methodology, G.D.; software, G.D.; validation, G.D. and A.L.; writing—original draft preparation, G.D.; writing—review and editing, G.D.; visualization, G.D.; supervision, A.L.; project administration, A.L. and P.S.; funding acquisition, P.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Italian Ministry of Education and University (MIUR) under the program PON R&I 2014-2020, grant number ARS01_01031.

Acknowledgments: The authors would like to thank colleagues of Masmec SpA for their support.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Atamuradov, V., Medjaher, K., Dersin, P., Lamoureux, B., & Zerhouni, N. (2017). Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation. International Journal of Prognostics and Health Management, 8(060), 1-31.
2. Silvestri, L., Forcina, A., Introna, V., Santolamazza, A., & Cesarotti, V. (2020). Maintenance transformation through Industry 4.0 technologies: A systematic literature review. Computers in Industry, 123, 103335.
3. Kuo, C. J., Chien, C. F., & Chen, J. D. (2010). Manufacturing intelligence to exploit the value of production and tool data to reduce cycle time. IEEE Transactions on Automation Science and Engineering, 8(1), 103-111.
4. Chien, C. F., & Chen, C. C. (2020). Data-Driven Framework for Tool Health Monitoring and Maintenance Strategy for Smart Manufacturing. IEEE Transactions on Semiconductor Manufacturing, 33(4), 644-652.
5. Si, X. S., Wang, W., Hu, C. H., & Zhou, D. H. (2011). Remaining useful life estimation–a review on the statistical data driven approaches. European journal of operational research, 213(1), 1-14.
6. Liao, L., & Köttig, F. (2016). A hybrid framework combining data-driven and model-based methods for system remaining useful life prediction. Applied Soft Computing, 44, 191-199.
7. Wang, Y., Zhao, Y., & Addepalli, S. (2020). Remaining Useful Life Prediction using Deep Learning Approaches: A Review. Procedia Manufacturing, 49, 81-88.
8. Elattar, H. M., Elminir, H. K., & Riad, A. M. (2016). Prognostics: a literature review. Complex & Intelligent Systems, 2(2), 125-154.
9. Pham, H. T., Yang, B. S., & Nguyen, T. T. (2012). Machine performance degradation assessment and remaining useful life prediction using proportional hazard model and support vector machine. Mechanical Systems and Signal Processing, 32, 320-330.
10. Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8), 1798-1828.
11. Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., & Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. Mechanical Systems and Signal Processing, 115, 213-237.
12. Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. IEEE Access, 7, 53040-53065.
13. Pearson, K. (1901). On lines and planes of closest fit to systems ofpoints in space. Philos Mag Series 2(11):559–57223.
14. Hotelling, H. (1933). Analysis of a complex of statistical vari-ables into principal components. J Educ Psychol 24(6):417–441.
15. Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7(2), 179–188.
16. Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504–507.
17. Ma, J., Su, H., Zhao, W. L., & Liu, B. (2018). Predicting the remaining useful life of an aircraft engine using a stacked sparse autoencoder with multilayer self-learning. Complexity, 2018.
18. Sun, C., Ma, M., Zhao, Z., Tian, S., Yan, R., & Chen, X. (2018). Deep transfer learning based on sparse autoencoder for remaining useful life prediction of tool in manufacturing. IEEE transactions on industrial informatics, 15(4), 2416-2425.

19. Ren, L., Sun, Y., Cui, J., & Zhang, L. (2018). Bearing remaining useful life prediction based on deep autoencoder and deep neural networks. Journal of Manufacturing Systems, 48, 71-77.
20. Liao, L., Jin, W., & Pavel, R. (2016). Enhanced restricted Boltzmann machine with prognosability regularization for prognostics and health assessment. IEEE Transactions on Industrial Electronics, 63(11), 7076-7083.
21. Haris, M., Hasan, M. N., & Qin, S. (2021). Early and robust remaining useful life prediction of supercapacitors using BOHB optimized Deep Belief Network. Applied Energy, 286, 116541.
22. Jiao, R., Peng, K., Dong, J., & Zhang, C. (2020). Fault monitoring and remaining useful life prediction framework for multiple fault modes in prognostics. Reliability Engineering & System Safety, 203, 107028.
23. Ma, M., Sun, C., & Chen, X. (2017). Discriminative deep belief networks with ant colony optimization for health status assessment of machine. IEEE Transactions on Instrumentation and Measurement, 66(12), 3115-3125.
24. Zhang, C., Lim, P., Qin, A. K., & Tan, K. C. (2016). Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics. IEEE transactions on neural networks and learning systems, 28(10), 2306-2318.
25. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
26. Dhillon, A., & Verma, G. K. (2020). Convolutional neural network: a review of models, methodologies and applications to object detection. Progress in Artificial Intelligence, 9(2), 85-112.
27. Babu, G. S., Zhao, P., & Li, X. L. (2016, April). Deep convolutional neural network based regression approach for estimation of remaining useful life. In International conference on database systems for advanced applications (pp. 214-228). Springer, Cham.
28. Heimes, F. O. (2008, October). Recurrent neural networks for remaining useful life estimation. In 2008 international conference on prognostics and health management (pp. 1-6). IEEE.
29. Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017, June). Long short-term memory network for remaining useful life estimation. In 2017 IEEE international conference on prognostics and health management (ICPHM) (pp. 88-95). IEEE.
30. Li, X., Ding, Q., & Sun, J. Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. Reliability Engineering & System Safety, 172, 1-11.
31. Malhotra, P., TV, V., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Multi-sensor prognostics using an unsupervised health index based on LSTM encoder-decoder. arXiv preprint arXiv:1608.06154.
32. Wu, X., Liu, Y., Zhou, X., & Mou, A. (2019). Automatic identification of tool wear based on convolutional neural network in face milling process. Sensors, 19(18), 3817.
33. Marei, M., El Zaatari, S., & Li, W. (2021). Transfer learning enabled convolutional neural networks for estimating health state of cutting tools. Robotics and Computer-Integrated Manufacturing, 71, 102145.
34. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.
35. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3), 211-252.
36. Krizhevsky, A., Nair, V., & Hinton, G. (2009). Cifar-10 and cifar-100 datasets. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Last accessed: 30-Jun-2021].
37. Mo, H., Custode, L. L., & Iacca, G. (2021). Evolutionary neural architecture search for remaining useful life prediction. Applied Soft Computing, 108, 107474.
38. Li, X., Jia, X., Wang, Y., Yang, S., Zhao, H., & Lee, J. (2020). Industrial remaining useful life prediction by partial observation using deep learning with supervised attention. IEEE/ASME Transactions on Mechatronics, 25(5), 2241-2251.
39. LMI Technologies. Gocator 3210 Datasheet - Large Field of View 3D Snapshot Sensor. [Online]. Available: https://lmi3d.com/resource/gocator-3210-datasheet-large-field-view-3d-snapshot-sensor/. [Last accessed: 14-Jul-2021].
40. Besl, PJ., McKay, ND. (1992). A Method for Registration of 3-D Shapes. IEEE Transactions on pattern analysis and machine intelligence. Los Alamitos, CA: IEEE Computer Society. Vol. 14, Issue 2, 1992, pp. 239-256.
41. Chen, Y., Medioni, G. (1992). Object modelling by registration of multiple range images. Image and vision computing, 10(3), 145-155.
42. Rousseeuw, P. J., Leroy, A. M. (2005). Robust regression and outlier detection (Vol. 589). John wiley & sons.
43. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1992, July). Surface reconstruction from unorganized points. In Proceedings of the 19th annual conference on computer graphics and interactive techniques (pp. 71-78).
44. LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10).
45. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.
46. Zinkevich, M., Weimer, M., Smola, A. J., & Li, L. (2010, December). Parallelized stochastic gradient descent. In NIPS (Vol. 4, No. 1, p. 4).
47. Ribani, R., & Marengoni, M. (2019, October). A survey of transfer learning for convolutional neural networks. In 2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T) (pp. 47-57). IEEE.
48. Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360.

49. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
50. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).
51. Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).
52. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).
53. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
54. Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).
55. Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-first AAAI conference on artificial intelligence.
56. Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).
57. Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 8697-8710).
58. Redmon, J. (2013). Darknet: Open source neural networks in c. [Online]. Available: https://pjreddie.com/darknet/. [Last accessed: 23-Jul-2021].
59. Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.
60. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.
61. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
62. Mitchell, M. (1998). An introduction to genetic algorithms. MIT press.
63. Beale, M., Hagan, M., & Demuth, H. (2021). Deep Learning Toolbox™ Reference. MATLAB (r) R2021a. The MathWorks, Inc. [Online]. Available: https://it.mathworks.com/help/pdf_doc/deeplearning/nnet_ref.pdf. [Last accessed: 27-Jul-2021].
64. Optimization Toolbox™ User's Guide. MATLAB (r) R2021a. The MathWorks, Inc. [Online]. Available: https://it.mathworks.com/help/pdf_doc/optim/optim.pdf. [Last accessed: 27-Jul-2021].
65. Cao, L. J., Chua, K. S., Chong, W. K., Lee, H. P., & Gu, Q. M. (2003). A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine. Neurocomputing, 55(1-2), 321-336.
66. Vapnik, V. The Nature of Statistical Learning Theory. Springer, New York, 1995.
67. Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. Statistics and computing, 14(3), 199-222.