Estimation of spot weld design parameters using deep learning

Akhil Pillai[1], Uwe Reuter[1], Marko Thiele[2]

TU Dresden[1]
SCALE GmbH[2]

1 Introduction

1.1 Motivation

In automotive production, each automobile has approximately 7,000 to 12,000 spot welds along with other kinds of connections. The position of the spot weld with respect to the flange and the distance between the spot welds as well as various other parameters usually vary for each part combination (spot weld design). If these properties are known, they can be used for automatic generation of spot welds during the design phase of the product development which is otherwise a cumbersome manual process. The spot weld design to be determined by the engineer depends on many factors (input parameters) such as loads and forces that might be applied to the structure, material combination, geometry of the parts, connection technology and its process parameters. Some of these parameters such as material combination and geometry of the parts are predefined by the designer or are results of the circumstances such as loads and forces applied at the connection. The remaining parameters such as connection technology, process parameters, spot weld distances and flange distance have to be chosen by the engineer. On the basis of existing designs and with help of machine learning techniques it may be possible to predict the spot weld design parameters like spot weld distance and flange distance. Within this work existing spot-weld designs are extracted from a vast amount of FEM simulation input data available in the Simulation Data Management (SDM) system LoCo of SCALE GmbH and applied as the basis for training and benchmarking new methods for estimating spot weld parameters.

1.2 Objective

In this work, the distances between spot welds are estimated for spot weld design using machine learning approaches. For a machine learning approach, the first step is to collect relevant data required to predict the desired outcome. Within this work, the desired outcome is the spot weld design parameter of distance between the spot welds. In order to predict the desired outcome, we make use of simulation data used for crash analysis. From the simulation data for crash analysis, we extract the geometry of the parts and also the spot weld designs. From the data of already developed car models, we can build a model which should be able to provide a good initial estimate for distance between spot welds. With this model the design engineer has a good initial estimate of distance between spot welds for different part combinations and hence the number of design iterations required to reach the optimum values decreases.

2 Mathematical basics

This chapter presents the mathematical concepts used in this work in a brief manner to provide basic understanding to the reader. The subsections cover the basics of Artificial Neural Networks (ANN's) and deep learning architectures.

2.1 Artificial neural networks

Basic explanations of Artificial Neural Networks (ANNs), on which this subsection is based, can be found in [1,2,3]. The formation of ANNs is an attempt to mathematically model the performance and intuitive capabilities of the human brain. The functionality of the human brain is essentially based on the interaction between the brain's highly cross-linked nerve cells called natural neurons. The communication within a natural neural network takes place via signals. A neuron serves for receiving, processing and passing on incoming signals. The signals received from neighboring nerve cells are summed in the neuron and if this simulation exceeds a particular threshold value, then further signal is activated and transmitted to the adjoining neurons.

This basic structure is imitated by ANNs. The ANNs are then used to realize complex mappings of input variables on output variables. ANNs mainly consist of cross-linked computational nodes or artificial neurons and communication takes place via numerical values. An artificial neuron receives numerical values from neighboring neurons (input neurons), which are combined to form a weighted sum. This determined sum is then compared with a threshold value (bias) and used as the argument for so called activation functions. The activation function yields a value (output signal) which is an input signal for further connected artificial neurons. An important type of artificial neural network is the multilayer perceptron. The artificial neurons are arranged in layers. Starting from an input layer, numerical values are transferred or propagated to an output layer via one or more hidden layers. The output layer provides the result for the corresponding input data. The input and output data are usually real numbers.

2.2 Convolutional neural networks

Convolutional networks [4], also known as convolutional neural networks, or CNN's, are a specialized kind of artificial neural network for processing data that have a known grid like topology. Examples of data on which CNN's can be employed include time series data, which can be thought of as 1-D taking samples at regular time intervals, and image data which can be thought of as a 2-d grid of pixels. The name "convolutional neural networks" indicates that the network employs a mathematical operation called convolution. Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of its layer. Research into convolutional network architecture proceeds so rapidly that we have a new best architecture for a given benchmark announced every few months. Hence it is impractical to describe the best architecture. However, all the architectures have mainly been composed of the building blocks described in this section. A typical layer of a convolutional network consists of three stages as shown in Figure (1). In the first step, a layer performs several convolutions in parallel to produce a set of linear activation's. In the second step, individual linear activation is run through a nonlinear activation function, mostly the rectified linear activation function. This step is sometimes called the detector layer. In the third step, we use a pooling function to modify the output of the layer further.

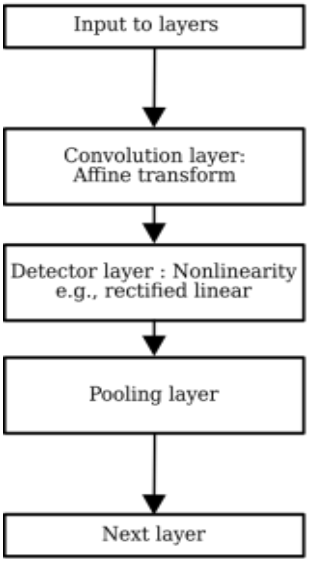


Fig.1: Components of typical convolutional neural network layer

2.3 Barycentric coordinates

In geometry, the barycentric coordinate system is a coordinate system in which the location of a point of a simplex (a triangle, tetrahedron, etc.) is specified as the center of mass, or barycenter, of usually unequal masses placed at its vertices. The system was introduced in 1827 by August Ferdinand Möbius [5]. Consider a set of points $P_0, P_1, \mathrm{K}, P_n$ and consider the set of all affine combinations taken from these points, i.e., all points P can be written as

$$\alpha_0 P_0 + \alpha_1 P_1 + \mathrm{L} + \alpha_n P_n \tag{1}$$

for some

$$\alpha_0 + \alpha_1 + \mathrm{L} \ + \alpha_n = 1 \tag{2}$$

Then this set of points forms an affine space, and the coordinates $(\alpha_0, \alpha_1, \mathrm{K}, \alpha_n)$ are called the barycentric coordinates of the points of the space [6]. These coordinates system is frequently useful and extensively used in working with triangles. This barycentric parameterization is exactly the parameterization that is usually used in our case for generating 3D geometric data.

In context of this work, we will make use of barycentric coordinates to generate points in a triangle. Consider three points P_1, P_2, P_3 in the plane and $\alpha_1, \alpha_2, \alpha_3$ are scalars such that $\alpha_1 + \alpha_2 + \alpha_3 = 1$, then the point P defined by

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 \tag{3}$$

is a point on the plane of the triangle formed by P_1, P_2, P_3. This point is within the triangle $\Delta P_1 P_2 P_3$ if

$$0 \le \alpha_1, \alpha_2, \alpha_3 \le 1 \tag{4}$$

If any of the α's is less than zero or greater than one, then the point P is outside the triangle. If any of the α's is zero then P is on one of the lines joining the vertices of the triangle.

In this work, we would utilize the concept of Barycentric coordinates to generate points inside elements of a part. In order to achieve this we would need to sample combinations of $\alpha_1, \alpha_2, \alpha_3$ so that we can generate points inside elements that are uniformly distributed. In order to do, we used the concept of Latin Hypercube sampling.

3 Machine learning approaches for classification of geometric data

3.1 Theoretical aspects of 3D geometric data

The raw 3D data that are captured by different methods come in forms that vary in both, the structure and properties. This section presents comprehensive information on different representations of 3D data by categorizing them into two main families:

Euclidean-structured data

Certain 3D representations have an underlying Euclidean-structure where the properties of the grid-structured are preserved such as having a global parameterization and a common system of coordinates. The main 3D representations that fall under this category are:

- descriptors
- 3D data projections
- RGB-D data
- volumetric data and
- multiview data

Non-Euclidean data

Another type of 3D data representations is the non-Euclidean data. This type of data representations suffers from the absence of global parameterization and the non-existence of a common system of coordinates or vector space structures [7], which makes processing of such data representations a challenging task. Considerable efforts are directed towards learning from such data and applying machine learning techniques on it. The main type of non-Euclidean data is point clouds, 3D meshes and graphs. Usually processing such data types happens on a global scale to learn the whole 3D object's feature which is convenient for complex tasks such as recognition and correspondence.

3.2 Deep learning on 3D data

3D data has multiple popular representations as described briefly in Section (3.1), leading to various approaches for learning. In this section we will briefly discuss some of the significant machine learning approaches applied to them.

- *Volumetric* CNNs: [8,9,10] are the pioneers applying 3D convolutional neural networks on voxelized shapes. Volumetric representation is constrained by its resolution due to data sparsity and computation cost of 3D convolution. FPNN [11] and Vote3D [12] has proposed methods to deal with the sparsity problem; however, it's challenging for them to process very large point clouds.
- *Multi-view* CNNs: [10, 13] have tried to render 3D shapes into 2D images and then apply 2D convolutional networks to classify them. This approach has achieved dominating performance on shape classification and retrieval tasks due to the fact that image CNN's are well engineered for classification tasks [14]. However the question of how many views are required to model the 3D shapes is still open.
- *Spectral* CNNs: Some of the latest approaches [15, 16] use spectral CNNs on meshes. However, these methods are currently constrained on manifold meshes such as organics objects and it's not obvious how to extend them to non-isometric shapes.
- *Feature-based* DNNs: [17, 18] extracts traditional shape features from 3D data and converts it into a vector. Then they use a fully connected network to classify the shape. This approach is constrained by the representation power of the features extracted.

3.3 PointNet

In this work, we explore deep learning architectures capable of reasoning about 3D geometric data of point clouds. Point clouds are simple and unified structures that avoid the combinatorial irregularities and complexities of meshes and are thus easier to learn from. The network named PointNet [19], provides a unified architecture for applications ranging from object classification, part segmentation, to scene parsing. Though simple, PointNet is highly efficient and effective. Empirically, it shows strong performance on par or even better than state of the art. In this section we will discuss about the approach of using this deep learning architecture for geometry classification.

PointNet is a unified architecture that directly takes point clouds as input and outputs either class labels for the entire input or per point segment/part labels for each point of the input. The basic architecture of the network is surprisingly simple as in the initial stages each point is processed identically and independently. A point cloud is represented as a set of 3D points $\{P_i \mid i = 1, \mathrm{K}, n\}$ where each point P_i is a vector of its (x, y, z) coordinate plus extra feature channels such as color, normal etc. For simplicity and clarity, we only use the (x, y, z) coordinates as our input. The network architecture, visualized in Figure (3), provides k scores for all the k candidate classes. The network has three key modules: the max pooling layer as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features.

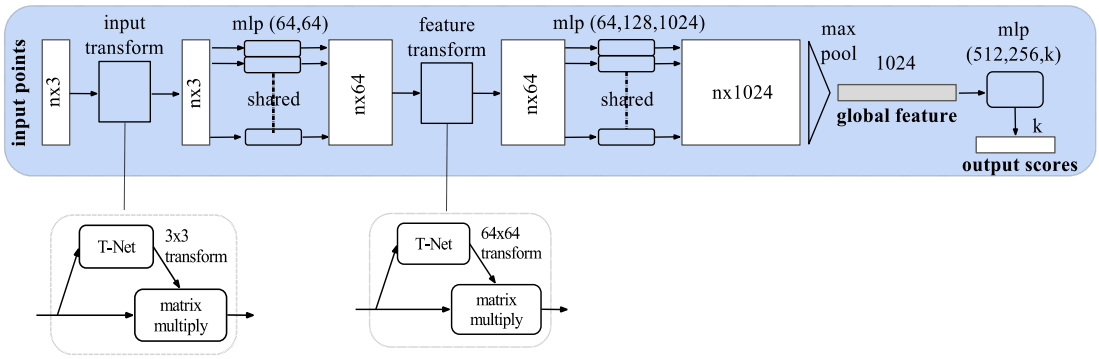


Fig.2: ***PointNet Architecture****. The classification takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes.*

The PointNet Architecture is implemented in TensorFlow, which is a open source machine learning frame work. In this work we used Python 2.7 to generate training data and Python 3.5 to train the PointNet architecture [20]. In order to speed up the training cycle we used a GPU (Graphics Processor Unit). TensorFlow allows us to use GPU's to train machine learning algorithms with the aid of CUDA and cuDNN. CUDA is parallel computing platform and programming model invented by NVIDIA. It enables dramatic increase in computing performance by harnessing the power of GPU [21]. The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU- accelerated library of primitives for

deep neural networks [22]. cuDNN provides highly tuned implementations for standard routines such as forward and back ward convolution, pooling, normalization, and activation layers. In this work we utilized CUDA 9.0 and cuDNN 7.0 for training purpose.

The training data for PointNet was stored in HDF5 binary data format. Python package “h5py” was used to achieve this. HDF5 is a high-performance data management and storage suite. It supports n-dimensional datasets and each element in the data set can itself be complex object. It comes with a set of integrated performance features that allow for access time and storage space optimizations. Also there is no limit on the number or size of data objects in the collection, giving great flexibility for big data. In this work we would be working with storing huge amount of geometric data as point clouds and hence HDF5 was chosen.

3.4 Generation of 3D geometric data from FEM data

In the previous section, a deep learning architecture is explained briefly which consumes point clouds and can be used to classify 3D geometry. In this section ideas and algorithms used to generate point clouds of 3D geometry for parts of car. Firstly we will have a look at how part geometry is expressed in FEM data. Then we will discuss methods to extract part geometry for individual parts and then using this part geometry data we will generate point clouds. The approaches presented were developed using libraries developed by SCALE GmbH.

3.4.1 Extraction of geometry from FEM data

In this subsection, we will discuss the ideas used to extract geometry data from FEM data. Nodes are the base for expressing geometric data in FEM data. Elements are formed with nodes and using elements we express part geometry. So when we want to extract geometric data, it indicates that we are extracting the coordinates of nodes for elements in a part. Now one can argue that we can just extract data of nodes present in a part and use it as point cloud of a part. This would be a simple process but we know that for feeding this point cloud into PointNet we would require the number of points in point cloud for each part to be the same. In Body in white (BIW) of a car not all parts are of same size and hence the number of nodes used to express the geometries of different parts will be obviously different. Hence it becomes necessary to extract data associated with elements that make up the parts and then use this data to generate points on surface of the parts according to our need. In this work we will only encounter with planar shell elements and thus the further discussions on elements will consider only the properties of shell elements.

SCALE GmbH has developed a python library called **femparser** with which one could parse the include decks of FEM data and extract geometric data. The output of **femparser** is a python object which can be used to extract the geometric data of individual parts using combination of part numbers, elements ID’s and node ID’s. The extraction algorithm is presented in Algorithm (1). The output is stored in JSON format files for the ease of reading the outputs and for further processing steps (i.e. generation of point clouds)

Algorithm 1 Part geometry extraction

Input: FEM include file of car
Apply femparser → mesh (python object)
procedure 1. Extract all part id’s
    for part in mesh **do**
        extract part id’s
    end for
end procedure
procedure 2. Extract geometry of each part
    for id in parts id’s **do**
        Extract element id from mesh
        for element id **do**
            extract nodal coordinates from mesh
        end for
    end for
end procedure
Output: Part JSON files

3.4.2 Generation of point clouds

In the previous subsection, we extracted the geometric data for individual parts and saved in JSON format. In this subsection we will discuss the approaches and algorithms used to generate point clouds for parts which will be used to train the PointNet. We have the coordinates of shell and triangular elements which make up the parts. To generate the point cloud for the complete geometry of the part we would have to generate points on the surface of elements. It is necessary to have a roughly equal distribution of points on the surface of the part so that concentration of points in certain regions does not affect the quality of results which we would obtain in further sections. We also have to take into consideration the fact that the number of elements which describe a part are not all equal. Also the size of parts varies according to their function. So we need to develop methods to generate point clouds which have points uniformly distributed across the part surface and capture the complete geometric information. In this work, two step approach is used. First we select elements so that we can capture the complete geometric information and then we triangulate the elements to generate points on surface using the concept of barycentric coordinates

Element Selection

First we have to decide the number of point's n_p to be generated in the point cloud. Once we know how many points to generate, we can employ different approaches to select elements, on the surface of which points will be generated. The simplest approach would be to generate one point, in random position, per element. Even though it sounds logical, it fails to generate equal number of points for different parts because they are not defined by the same number of elements. The next approach is called the naive random selection technique, in which we can select randomly n_p elements and generate one point, in a random position, inside the corresponding element. This approach looks fairly simple and would provide us a different point clouds for same part. The problem with this approach is that we cannot guarantee a fairly equal distribution of points over the complete part since elements are chosen randomly. Also when the number of points n_p is greater than the number of elements available for point generation, some elements can be chosen multiple times and this would lead to concentration of points at certain areas which are not under our control. In certain cases this would also lead to loss of geometric information.

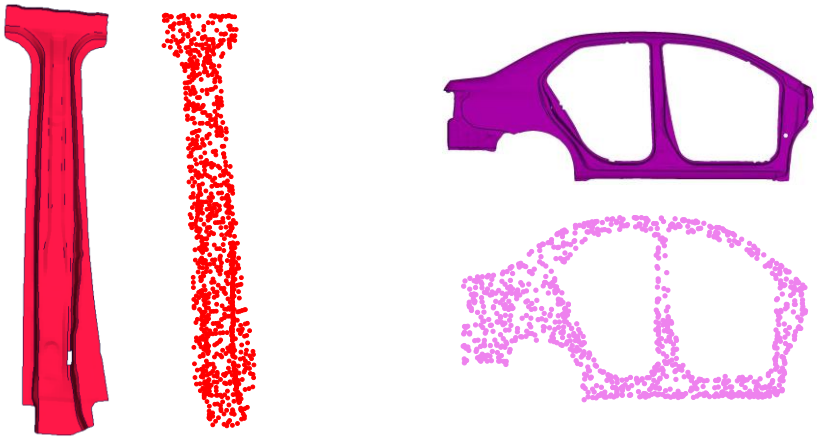
In order to improve the sampling of elements, we could use the area of element as a parameter. Using area of element as a parameter we can estimate the significance of an element. When the number of elements is greater than n_p, we would want to generate more points in elements with larger area. With this we can also mitigate the problem of loss of geometric information when the number of points n_p is less than number of elements in a part. In this work, algorithms were developed considering the above mentioned aspects. The areas of elements were used as a parameter to select the significant elements and weights were generated to estimate the number of points to be generated within the element using Equation (5) where A_i is the area of the i^{th} element and n is the number of elements which defines the part.

$$w_i = \frac{A_i}{\sum_{i=1}^{n} A_i} \times n_p \tag{5}$$

Point generation

Now that we know the number of points to generate on surface of each element of a part, the next step is to discuss the method used for generating point inside the element. In Section 2, we discussed the concept of barycentric coordinates and Latin hyper cube sampling technique. These concepts will be used for generating points on the surface of elements of the parts. The concept of barycentric coordinates can be applied directly to triangular elements, but in our case we also have rectangular elements. The easiest way to generate point for rectangular elements would be triangulate them and then use the concept of barycentric coordinates to generate points on their surface. This method is employed in this work for generating points on surface for rectangular elements. Figure (3) shows the

examples of point clouds of some of the parts of 2010 Toyota Yaris model which is detailed Finite Element Model available in public domain [23].



(a) B-pillar *(b) side panel*

Fig.3: Point clouds of parts of 2010 Toyota Yaris model

3.5 Part identification of car model using PointNet

3.5.1 Toyota Yaris Model

In this subsection we will try to classify all the parts present in BIW of the TOYOTA YARIS model (Figure 4). 250 different parts were identified and extracted from the FEM model. We have not combined mirrored parts i.e. part with identical geometry on left and right side of the car. It is necessary for us to know if PointNet is able to identify them with respect to their positions. The PointNet was trained on Nvidia GeForce GTX 750 Ti for n_p = 1024 with the following input parameters for the architecture

- batch size = 10
- learning rate = 0.001
- momentum = 0.9
- optimizer = ADAM
- number of samples in training set = 10, 000
- number of samples in test set = 2500
- number of epochs = 300

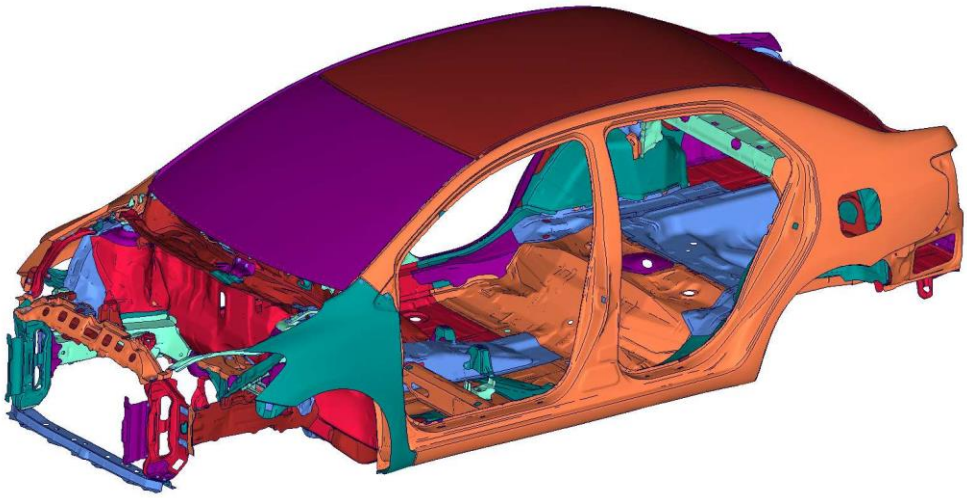


Fig.4: BIW of Toyota Yaris Model

Figure (5) shows the performance plots of PointNet when trained for 250 parts of the Yaris model. The training time for 300 epochs is ~ 30 hours. PointNet was able to achieve 88% training accuracy. The trained model was then used to predict the class labels for unseen sample of point cloud for 250 parts. The model was able to predict 237 class labels correctly for 250 parts. With these results it becomes evident that 95% of part labels are predicted accurately by PointNet for the Yaris model. It is also interesting to note that the model can accurately differentiate similar geometries located on either side of axis of symmetry of the car. The only downside here is we had only one version for each part. It would be interesting to see the performance of PointNet when we have different versions of same part. In order to do so we would require information of different versions of same part.

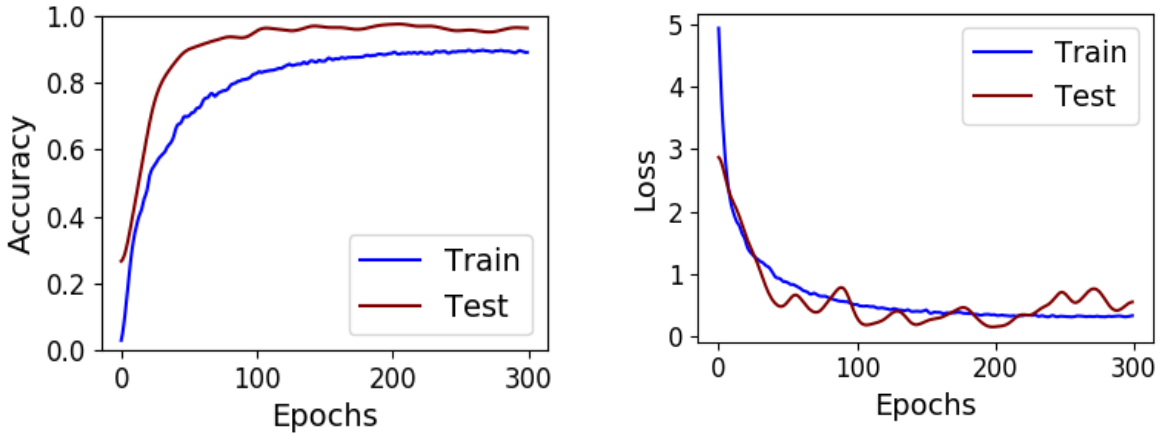


Fig.5: Performance of PointNet for Toyota Yaris Model

3.5.2 AUDI model

The Simulation Data Management System LoCo provided an example of AUDI model where we had the opportunity to test the performance of PointNet when different versions of same parts exist. LoCo had information of a FEM model of an AUDI car for a whole development cycle of roughly 5 years. From this data we used 2 include files to check if some parts have different versions and we found 13 parts with different geometries. The model consisted of 350 parts and PointNet was trained with the following input parameters for the architecture for n_p = 1024.

- batch size = 10
- learning rate = 0.001
- momentum = 0.9
- optimizer = ADAM
- number of samples in training set = 17, 500
- number of samples in test set = 5000
- number of epochs = 25

Figure (6) shows the performance plots of PointNet for the AUDI model. PointNet was able to achieve 80% training accuracy and the training time was ~ 48 hours. On evaluation of the trained model with new set of point clouds, the model was able to identify 339 out of 350 parts correctly. This means that the trained model predicted 96% of the part labels accurately. These results are quite promising.

The trained model was then used to predict the part label for new versions of parts used for training the model. Figure (7) shows an example of part version where the part has some changes. The model was trained to identify the geometry shown in Figure (7a). The trained model was used predict the part label of a version of the same part where the part has a difference as shown in Figure (7b). In reality the model has not seen this new version in its training data but the trained model is able to predict the correct part label. One could argue that the sampling methods used to generate the point clouds would not have captured such a small change and thus for the trained model it doesn't make any difference. In reality this would not be the case as the PointNet uses overall geometry and the position of the part also as information for predicting class label. The trained model was also able to predict accurate class labels for remaining 11 parts with versions. With this example we could see that the

trained network is able to predict correct class labels even when the part geometry changes but still maintains positional similarity and some geometrical similarity to the part used for training the model.

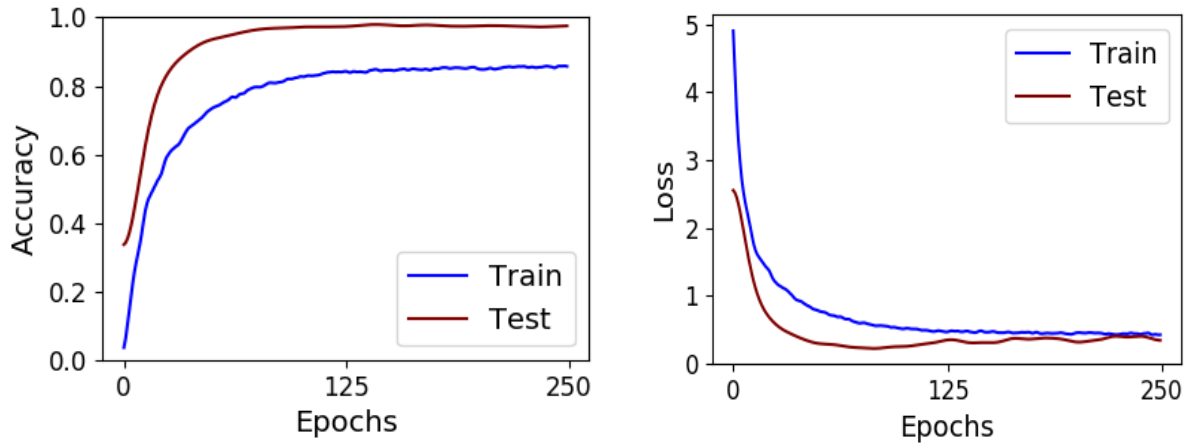
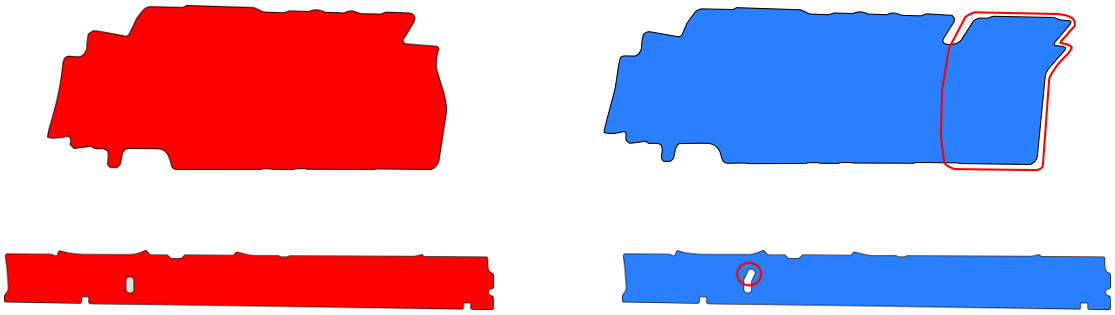


Fig.6: Performance of PointNet on AUDI model



(a) Training geometry (b) New version

Fig.7: Examples of part versions

3.6 Part combination identification using PointNet

This section presents the investigations conducted to identify various part combinations of the AUDI model. Using the ideas presented in Section 4.1, we identified 873 part combinations in AUDI model connected using spot weld. We want to investigate if we can extend the PointNet model to consume point cloud of part combinations and provide an accurate identification of the part combination. The basic idea is to give each part combination a unique label and evaluate if the architecture is able to provide accurate prediction of class label for all the part combinations. In this case also the position of the part combination is important geometric information for identification of the part combination.

The PointNet architecture was trained for n_p = 1024 with the following input parameters:

- batch size = 10
- learning rate = 0.001
- momentum = 0.9
- optimizer = ADAM
- number of samples in training set = 43, 650
- number of samples in test set = 5000
- number of epochs = 300

The performance plots are show in Figure (8). The evaluation accuracy is just 60% i.e., the trained network is able to predict only 524 part combination labels correctly. This kind of performance is not useful for our further works. On inspection of predicted class label for part combinations, the trained model is unable to differentiate part combinations when one part in the combination is significantly

larger in size than the other part. This leads to masking of the smaller geometry by the bigger geometry which PointNet is not able to distinguish.

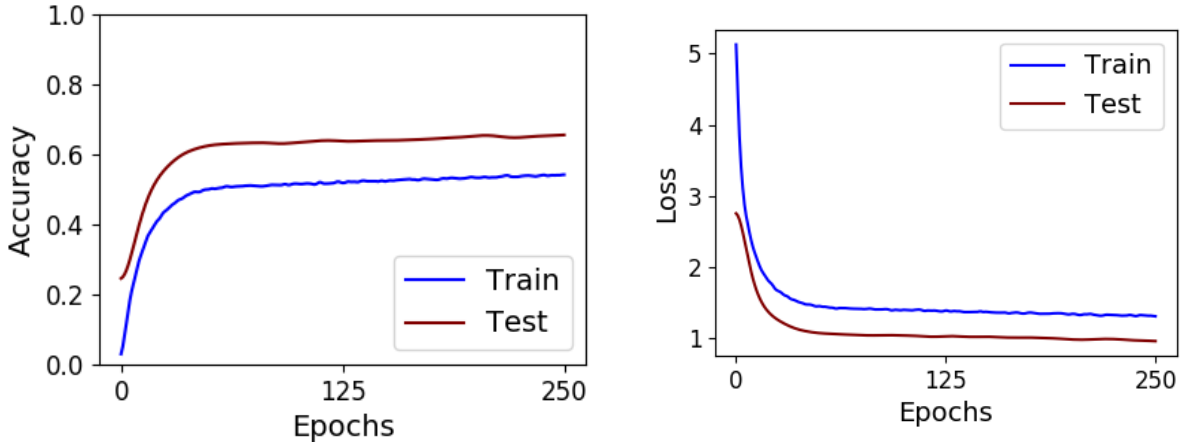


Fig.8: Performance of PointNet on part combinations of AUDI model

4 Machine learning approaches for estimation of spot weld design

4.1 Spot weld design parameter

The spot weld design, determined by the design engineer depends on many factors, i.e., input parameters like loads and forces that might be applied to the structure, material combination and geometry of the parts, the connection technology and its process parameters. Our focus is mainly on parameters like position of the spot weld with respect to the flange and the distance between the spot welds. Both these parameters usually vary for each part combination. In this section we will describe the methods use to calculate these parameter of distance between spot weld from FEM model.

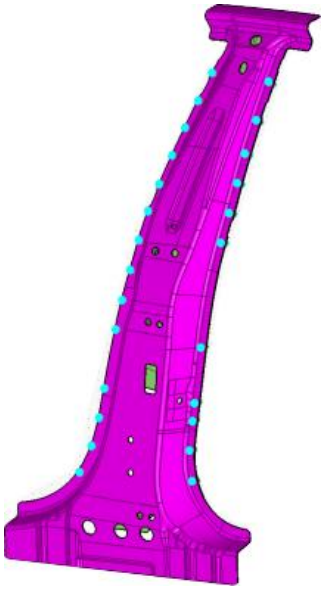


Fig. 9: B - pillar

Figure (10) shows an example of *b-pillar* of a car with spot weld design. The spot weld design obtained for this example is a final result of numerous design and engineering iterations. The design varies with respect to materials used and the sheet thickness of the part. As an engineer our target would be to use machine learning for prediction of this spot weld design without the effort and cost of numerous engineering simulations. As this is a “first of its kind” work, initially we would be working towards estimation of minimum distance between spot welds so that we a have a good starting design which would need to go through minimum number of design iterations. In doing so we would be at least saving a significant amount of cost and computational efforts required to reach optimum spot weld design.

Consider we have n spot weld points for a part combination. Algorithm (2) presents the method employed to estimate d_s for a part combination. There exist many methods to find the closest point for a chosen point from a given set of points. In this work we employed a linear search solution, in which for a given point we computed the Euclidean distance to all other points and the closest point is the one which is at the shortest distance to the given point.

Algorithm 2 Calculate d_s for a part combination

Input: P_1, K, P_n

for $i = 1, \mathrm{K}, n$ **do**

Find closest point P_i^c for P_i

Calculate Euclidean distance between P_i^c and P_i

end for

Output: $d_s = \dfrac{\sum_{i=1}^{n} d_i^c}{n}$

4.2 Estimation of minimum distance between spot welds

4.2.1 Classification based approach

Classification based approach considers the use of PointNet to estimate minimum distance between spot welds. In Section 3.6 we discussed the idea of using PointNet to identify part combinations and results obtained were poor. So instead of using PointNet to identify parts, we would try to generate class labels for d_s and train the PointNet to identify these class labels for part combinations.

In previous geometry identification experiments, we labeled each part combination with a unique label. However in this experiment we label the part combinations according to a criterion, which their corresponding d_s value satisfies. The basic idea is to use classifier to predict values for d_s based on part combinations. This approach was an attempt to use the geometry as an input to predict the parameter d_s. Table (1) shows the labeling criteria for part combinations based on d_s.

Criteria (mm)	label
$d_s = 0$	0
$0 < d_s < 2$	1
$2 < d_s < 4$	2
M	M
$98 < d_s < 100$	51
$d_s > 100$	52

Table 1: Labeling criteria for classification based approach

The PointNet was trained with the newly labeled training data of part combinations for n points = 1024 with the following parameters:

- batch size = 10
- learning rate = 0.001
- momentum = 0.9
- optimizer = ADAM
- number of samples in training set = 52, 380
- number of samples in test set = 873
- number of epochs = 200

From the performance plots in Figure (10), we can see that the evaluation accuracy of the model reached nearly 90% during training. The trained model was evaluated using a new set of point cloud for each part combination and the prediction accuracy was 86%. The trained model was able to predict the class labels of 758 part combinations correctly. The 115 wrong predictions by the trained model were analyzed. For 56 part combinations the predicted class label was less than the actual class label and for 59 class labels the predictions were greater than actual class label.

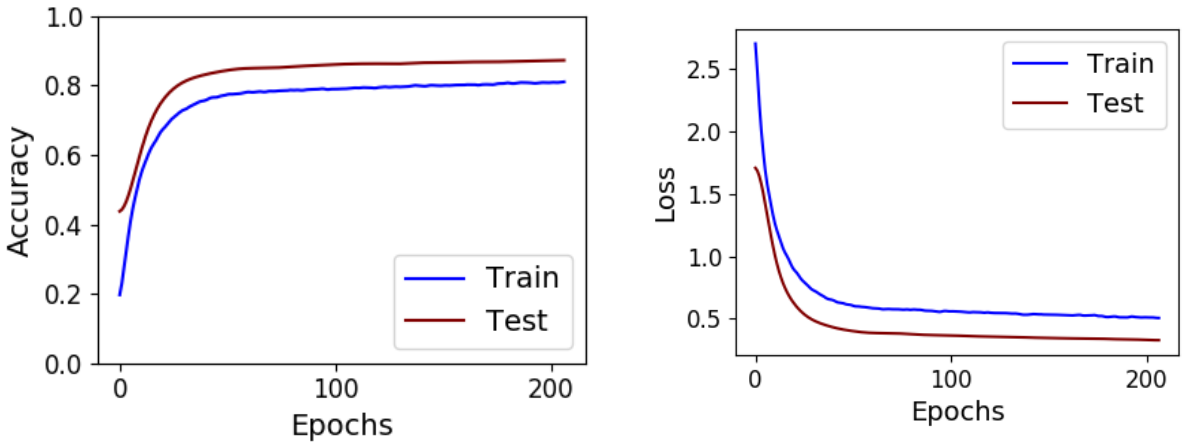


Fig.10: Performance of PointNet for classification based approach

4.2.2 Prediction based approach

The basic idea of this approach was to use the identification capability of PointNet to identify the parts, and then use another neural network to estimate spot weld parameters. The approach is outlined in Figure (11). Here we first use PointNet to identify the parts from their point clouds and then use the part identification to predict spot weld parameters. PointNet is able to identify 96% part labels of AUDI model accurately. So in this approach we try to build a model which consumes part labels for two parts and predicts spot weld parameter d_s .

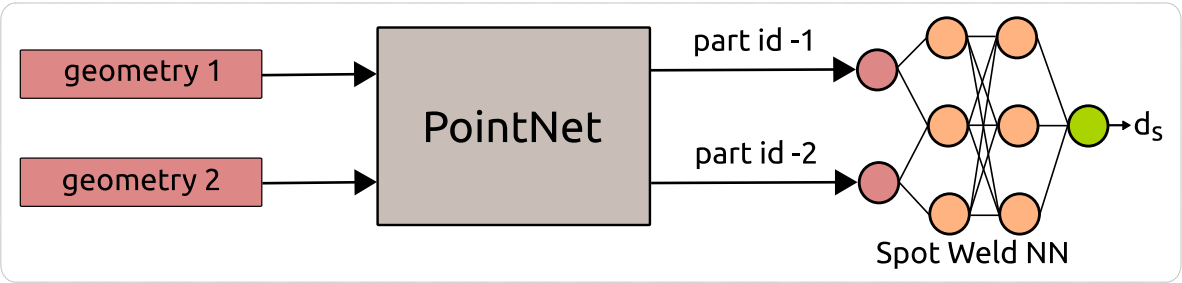


Fig.11: Prediction based approach

In this work we utilized a feed forward neural network with two hidden layers. The parameter prediction model is a non linear model and we would require at least two hidden layers to have meaningful results. The parameter prediction model was implemented using TensorFlow. The important aspect in this model was data preparation and the model architecture.

In order to train the spot weld parameter predictor model, a training data set was generated from the AUDI model. The part labels and the spot weld parameter d_s were normalized in the range [0, 1] for training the feed forward neural network. Since we did not have a bench marked architecture for this type of problem, many architecture were experimented upon and only the architecture which provided meaningful results has been presented in this work. A feed forward neural network with 150 neurons in the first hidden layer and 50 neurons in the second hidden layer was trained to predict the spot weld parameter d s and the results obtained are outlined in this work. Back propagation algorithm was used to train the model and gradient descent optimization was utilized with Mean Squared Error as the cost function. The model was trained with the following parameters:

- batch size = 10
- learning rate = 0.1
- number of samples in training set = 846
- number of epochs = 500, 000

Figure (12) shows error v/s epochs plot during model training. The trained model was then used to predict spot weld parameter for the AUDI model.

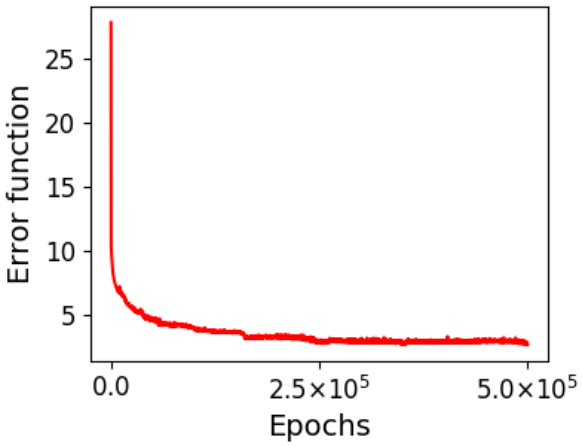


Fig.12: Training spot weld parameter predictor model

We calculated the deviation of the predicted values by the trained model so that we can have an estimate of the overall performance of the model. The deviation d_e was estimated using Eq. (6)

$$d_e = \text{Actual } d_s - \text{Predicted } d_s \tag{6}$$

Table (2) shows the performance of the trained model. The performance accuracy is calculated as the percentage of number of part combinations for which the d_e value satisfies the mentioned deviation criteria.

Deviation in mm	Performance accuracy
$d_e = 0$	30.02 %
$d_e < 2$	65.85 %
$d_e < 5$	80.73 %
$d_e < 10$	86.00%

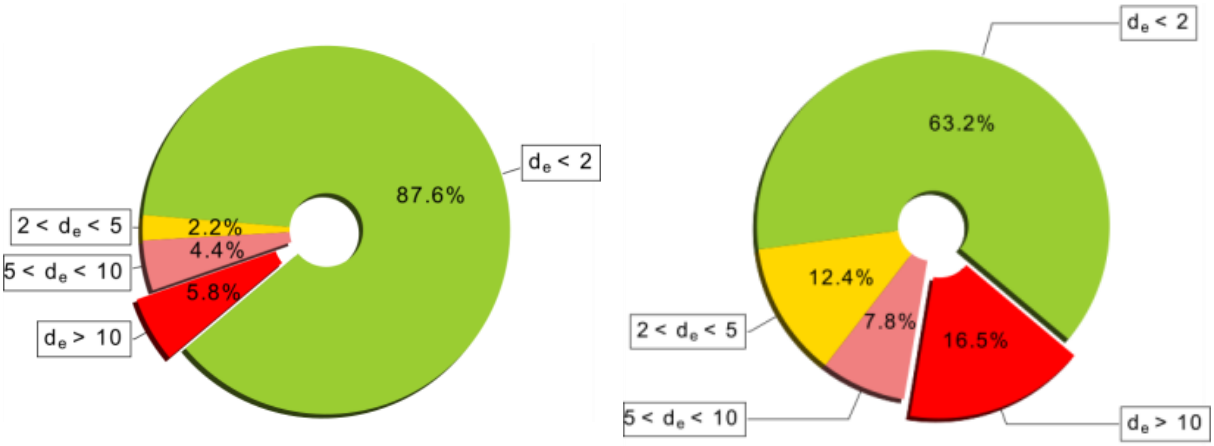
Table 2: Performance of spot weld predictor model

4.2.3 Comparison of approaches

In order to evaluate the performance of the approaches mentioned in previous subsections, a comparison study was performed. The methodologies used to predict spot weld parameter d_s are different for classification based approach and prediction based approach. Classification based approach directly consumes the point cloud of part combinations and provides a prediction of d_s within an interval of width 2 mm. Prediction based approach consumes point cloud of individual part in a part combination and provides a numeric estimation for d_s. In order to evaluate the performance of both approaches we calculate the deviation of predicted value of d_s. We calculate the deviation d_e for both approaches differently. d_e for classification based approach is estimated using Eq. (7)

$$d_e^{\text{Classification}} = \left(\text{Predicted } c_L - \text{Actual } c_L\right) * \text{Interval size} \tag{7}$$

where c_L is the class label and interval size is the difference between upper limit and lower limit of the criteria defining the class labels. For prediction based approach, we estimate d_e using Eq. (6). During evaluation of performance for prediction based approach, we also factor in the fact that identification of PointNet for all parts of AUDI model is not 100%. We identified the wrongly labeled parts and propagated this error in identification to the final prediction of d_s. The results obtained are visualized in Figure (14).



(a) Classification based approach *(b)* Prediction based approach

Fig.13: Comparison of approaches

In Figure (14), the results displayed are percentage of total number of part combinations analyzed. From the results, we can see that classification based approach provides better estimation for larger number of part combinations as compared to prediction based approach. But classification based approach doesn't provide us the possibility to include parameters like sheet metal thickness or material properties in prediction of spot weld parameters. Prediction based approach provides good results with the possibility of extending the model to include the parameters mentioned above in order to have more realistic results.

Classification based approach and prediction based approach have their own strengths and shortcomings. In classification based approach, we are directly making use of geometric information of the parts to predict spot weld parameters whereas in prediction based approach, the prediction of spot weld parameter has no relation to geometric information. Since this work is first of its kind, we are looking for methods which can incorporate more input parameters for predicting spot weld design, thereby increasing the credibility of prediction, and thus Prediction based approach looks promising.

In future works, the performance of prediction based approach can be improved by experimenting with different input parameters of neural network architecture. We can also conduct experiments on different architectures which can provide better performance when we consider more input parameters for prediction of spot weld design.

5 Future works

In this work, we presented the idea to identify parts of car model using a machine learning model directly from the geometric data and then use this information to predict spot weld design parameters for part combinations. In future work, we would want to expand this idea to use information of spot weld design in existing car models and then predict spot weld design parameters for a newly designed car model and also develop methods to generate spot weld designs automatically using the predicted parameters. This would mean that, the model would have to first identify an unseen part and then predict the spot weld parameters for this new design based on information from previous designs in older car models or other car models. This would have a considerable impact in reducing the product development cost and time.

6 Conclusion

In this work, we have developed and defined methods to extract significant input parameters of existing 3D geometry from FEM data. The extraction of significant parameters was completed with aid the library femparser and Python scripts. We also defined methods to generate point clouds of individual geometries from significant input parameters of existing 3D geometry. The identification of individual part using point clouds was achieved using the deep learning architecture called PointNet. The results of this classification problem were analyzed and we also tried to reason for wrong

predictions of classification problem. Due to the complexity of modeling parts in FEM, for some parts extraction of individual geometry was not feasible and this has contributed to minor errors in identification problem. In further works, we should be able to solve this problem.

In order to estimate parameters for spot weld design, we first extracted significant input parameter of minimum distance between the spot welds for existing FEM data. This process was also achieved with aid of femparser and python scripts. In this work, we have outlined two different approaches which were used to estimate spot weld parameter. Evaluation of performances of these two approaches was conducted and their results are also compared.

Classification based approach dealt with the idea of directly using geometric data to predict parameters. It involved the transformation of prediction problem into a classification problem for PointNet. Prediction based approach used PointNet first to identify the parts from their geometries and then used this information to provide a prediction for spot weld parameter using a separate neural network. It was found out that, in order to improve the credibility of estimation of parameters for spot weld design, Prediction based approach provided suitable base for further research.

With this work, we are able to provide credible results for identification of parts using point cloud of parts and PointNet. We also have outlined approaches with which we estimated minimum distance between spot welds for part combinations. These approaches can further be developed for estimation of further parameters of spot weld design.

7 References

[1] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[2] T. M. Mitchell, Machine Learning, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[3] D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations. Cambridge, MA, USA: MIT Press, 1986.

[4] Y. Lecun, Generalization and network design strategies. Elsevier,1989.

[5] E. Hille, Analytic Function Theory, 2nd ed. New York, NY, USA: Chelsea Publishing Company, 1982.

[6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond euclidean data,” IEEE Signal Processing Magazine, vol. 34, no. 4, pp. 18–42, July 2017.

[7] T. DeRose, “Coordinate-free geometric programming,” Seattle, Washingtion, Tech. Rep., 1994.

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[9] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep. 2015, pp. 922–928.

[10] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, “Volumetric and multi-view cnns for object classification on 3d data,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[11] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, “Fpnn: Field probing neural networks for 3d data,” in Advances in Neural Information Processing Systems 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 307–315. [Online]. Available: http://papers.nips.cc/paper/6416-fpnn-field-probing-neural-networks-for-3d-data.pdf

[12] D. Zeng Wang and I. Posner, “Voting for voting in online point cloud object detection,” in Proceedings of the Robotics: Science and Systems, Rome, Italy, 07 2015.

[13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” in The IEEE International Conference on Computer Vision (ICCV), December 2015.

[14] M. Savva, F. Yu, H. Su, A. Kanezaki, T. Furuya, R. Ohbuchi, Z. Zhou, R. Yu, S. Bai, X. Bai, M. Aono, A. Tatsuma, S. Thermos, A. Axenopoulos, G. T. Papadopoulos, P. Daras, X. Deng, Z. Lian, B. Li, H. Johan, Y. Lu, and S. Mk, “Large-scale 3d shape retrieval from shapenet core55: Shrec’17 track,” in Proceedings of the Workshop on 3D Object Retrieval, ser. 3Dor ’17. Goslar Germany, Germany: Eurographics Association, 2017, pp. 39–50. [Online]. Available: https://doi.org/10.2312/3dor.20171050

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” CoRR, vol. abs/1312.6203, 2013. [Online]. Available: http://arxiv.org/abs/1312.6203
[16] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, “Geodesic convolutional neural networks on riemannian manifolds,” in The IEEE International Conference on Computer Vision (ICCV) Workshops, December 2015.
[17] Y. Fang, J. Xie, G. Dai, M. Wang, F. Zhu, T. Xu, and E. Wong, “3d deep shape descriptor,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.
[18] K. Guo, D. Zou, and X. Chen, “3d mesh labeling via deep convolutional neural networks,” ACM Trans. Graph., vol. 35, no. 1, pp. 3:1–3:12, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2835487
[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” arXiv preprint arXiv:1612.00593, 2016.
[20] Q. Charles, “pointnet,” Available at https://github.com/charlesq34/pointnet (2019/02/22), 2017.
[21] NVIDIA, “Cuda,” Available at https://www.geforce.com/hardware/technology/cuda (2019/02/22).
[22] “cudnn,” Available at https://developer.nvidia.com/cudnn (2019/02/22).
[23] “2010 toyota yaris detailed finite element model,” Available at https://www.ccsa.gmu.edu/models/2010-toyota-yaris/ (2019/02/22), 2016.