Foundations of Smart Manufacturing, Spring 2021 Paper Series

Classification of 3D Objects Related to Mechanical Assemblies

Parth Mujumdar

Abstract

Neural networks (NNs) are used for forecasting, pattern recognition, optimization and decision making in various industry sectors like banking, supply chain, software development, healthcare, and manufacturing. Convolutional neural networks (CNNs) are extremely popular with computer vision(CV) tasks like image classification, object detection among the popular applications of CNN. The manufacturing industry can benefit many folds by investing in the ongoing artificial intelligence(AI) and machine learning (ML) revolution. This project report paper proposes an algorithm that successfully classifies manufacturing parts using data derived from 3D computer-aided design(CAD) models. This project report also explores the scope of multi-view convolutional neural networks (MVCNN) being used for more efficient classification of 3D CAD models. This study will help develop an efficient model for manufacturing part classification. Such models can be used in the industry for quick retrieval of CAD models from large public datasets, autonomous quality control and improving the generative design technology.

Keywords: multi-view convolution neural networks; computer vision; computer-aided design; big data analytics; industry 4.0

---

1. Introduction

The advent of Industry 4.0 has every machine spitting out data and the storage, management, and analytics of this data is a big challenge industry is facing right now. This problem has resulted in big data storage and analytical systems being developed at a rapid rate. Technologies like AI and ML are lending a hand to handle and use this data more efficiently. While software industry is at the forefront of implementation of these tools and technologies, industries like finance[1], healthcare[2] and entertainment are also catching up fast. Manufacturing industries, however, have seen a slow growth when it comes to being more data efficient. Research from Exasol, a German based database analytics firm, has found that 75 per cent of manufacturing business leader's operational decision-making would be completely or significantly disrupted without efficient use of data analytics[3]. Manufacturing industry has one of the highest data generation rates compared to any other industry, using this data to facilitate impactful changes is a challenge facing the industry. 3D CAD data is generated by every design team in every manufacturing organization. Facilitating ease of generation and classification of these models will save valuable time spent in CAD design search and retrieval. Neural networks can be used to accomplish these tasks.

Neural networks are a subset of machine learning and very popularly used in deep learning algorithms. The working of neural networks is comparable to the working of neurons in a human bring. Every neural network model is trained using training data to improve its accuracy. The trained model is then used on test data to check for its validity and accuracy. Every neural network architecture has three types of layers. The architecture of a NN can be seen in figure 1. Every neuron in a NN gets an input and a random weight is assigned to that input at the beginning of model training. The product of this input and random weight is then summed and fed to a nonlinear activation function. An activation function helps to decide whether a neuron should be activated or not[4]. The random weight is calibrated throughout the training cycle in a way such that the error in the output is minimum. The calibration of random weight takes place by using the gradient descent method. The working of a NN can be seen in the figure 2.

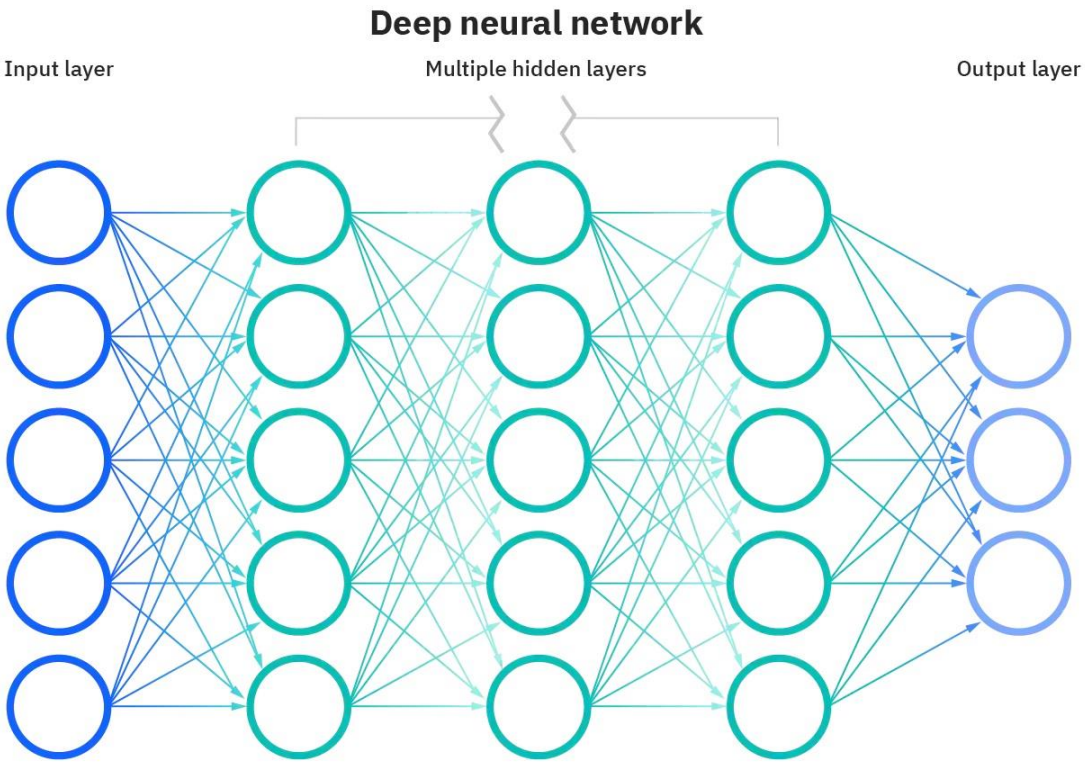

Figure 1 : Architecture of Neural Networks[5]

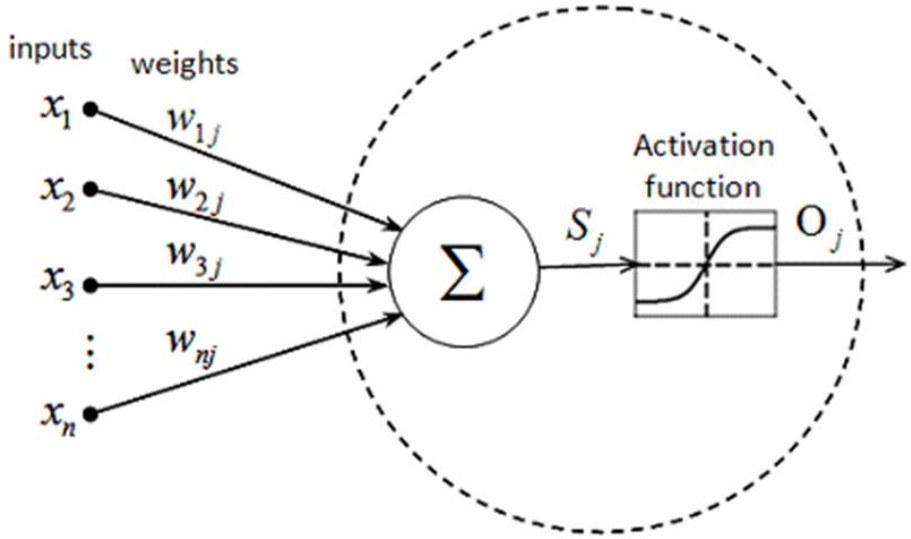

Figure 2 : Working of Neural Networks[6]

Convolutional neural networks are a type of NN that is very popular for use in computer vision tasks. It is important to understand how a computer machine reads an image before we dive into the working of CNN. Every pixel in an image is associated with a numerical value. This numerical value is the color code for that pixel. This results in the machine converting an image into a matrix of values(or an image matrix) where every value represents the color code for that individual pixel. It is very important to maintain the spatial arrangement of matrix in both horizontal and vertical direction. This matrix is then used as a representation of the input image for feature extraction. Convolutional neural networks use a filter to extract features from

the input image. The filter can be thought of as a weighted matrix that moves around the image matrix extracting its features.

The CNN can be though of as having three basic layers:

- Convolutional and/or pooling layer: The convolutional layer is the filter moving around the image matrix extracting its features and reducing the size of the matrix for processing. There can be multiple convolutional layers in a CNN model. Pooling layers are used to reduce the dimensions of trainable features of an image. This layer is used when the number of trainable parameters is too high in case of large images.
- Fully connected layer: Convolutional layers usually can't generate outputs equal to the number of categories required. A fully connected layer can do this job and is connected to the output layer.
- Output layer: The output layer gives the output of the model and uses a loss function to check for the accuracy of the model. This loss function helps to improve the prediction and bring it closer to the actual value.

Many different NN architectures have been used for the purpose classification of 3D objects. These models generally use the 3D shape files of the CAD objects for training purposes. Multi-view convolutional neural network (MVCNN) tries to build on the idea that, instead of using multiple data attributes of a 3D object for classification, if quality images showing different views of the object or 'multi-view' of the object are used, this will lead to a better performance of the model. In this study both the classification methods have been explore, one that uses 3D file as the input and one that uses a image of 3D file as the input.

2. Related Work

A lot of studies have been done on the classification of 3D models. Some of the early and prominent ones have been discussed in this section.

In the 'ShapeNet: An Information-Rich 3D Model Repository' study done by Angel X.Chang et el, they created a dataset called ShapeNet. The data for this dataset came from public online repositories or existing research datasets. This dataset contains more than 3,000,000 models of which 220,000 were classified into 3,135 categories as a part of this study[7]. The dataset provided many semantic annotations which were made available through public web-based interface[7]. The data used for building of ShapeNet dataset was organized in a hierarchical categorization. The data contains language related annotations, geometric annotations like rigid alignment, parts and key points, symmetry and object size, functional annotations like parts and affordance and physical annotations like surface material and weight. The study specially stresses on how these annotations make the ShapeNet study valuable. These annotations were described using a hybrid methodology. In a hybrid methodology, first, the annotation is algorithmically predicted and then undergoes a manual inspection under a human expert[7]. This dataset has been used by several research studies for training and building of 3D object classification models. A lot of readily available trained models have used this dataset.

The study performed by Zhirong Wu et el trains the 3D ShapeNet model on the ModelNet dataset, which is a large-scale 3D CAD model dataset. The 3D ShapeNet model discovers hierarchical compositional part representations automatically. The 3D ShapeNet study proposes a model to represent the geometric 3D shape as a probability distribution of binary variable on a 3D voxel grid[8]. Along with MVCNN, Convolutional Deep Belief Networks (CDBN) are also used to classify 3D voxel data from 2D pixel data. A simple DBN would have created a huge number of parameters which would have been difficult to train on. In such cases a convolutional layer without any pooling layers can help to reduce model parameters and make the data trainable [8]. This research paper also proposes 'Next-Best-View' Prediction which seems to be very similar to MVCNN, except it uses data from voxel grid rather than an image. This proposed method suggests supplementing the model with second different view of the same object which can help to reduce the recognition uncertainty. Given a first view to the model, the 3D ShapeNet model is also able to recognize the view that has a high probability of classifying the 3D model correctly. The model uses a linear SVM for

classifying the meshes and uses average category accuracy to evaluate the performance of the model[8].

The study performed by Hang Su et el[9] argues that the view-based descriptors can do a better job of recognizing 3D objects as compared to descriptors like voxel grid or mesh which operates on their 3D format. As a proof of concept, they initially trained a standard CNN to classify views of different objects. The study puts forward the view that training 3D classification models on different views of same objects can yield higher classification accuracy as compare to models using descriptors that extract features from a 3D file format. Another argument to use image-based descriptors was that with advancement in image quality the quality of data will also improve. Also, there are already image-based dataset like ImageNet present which can be used to pre-train the CNN models. This MVCNN model was evaluated on the Princeton ModelNet[8] dataset discussed above[9]. Their research also proposes a sketch-based 3D shape retrieval model. Along with the classification aspect of the model, they also test the model for retrieval of 3D objects, which is another use of classification model built on 3D object data.

The study done by Qian Yu et el explores the Latent MVCNN method to train models for 3D shape recognition. The paper argues that 3D shape recognition using small number of view-images is considerably difficult. Latent MVCNN uses three layers of convolutional networks which help the model to calculate category probability distribution for every pair of images fed to the model. This model was also trained on the ModelNet dataset. The LMVCNN model was not initially built for 3D object retrieval purposes. However, after testing the trained model to retrieve 3D object the results were promising considering the small number of view images.[10]

The study performed by Atin et el on the possible use of metadata to supplement the image-views used to train a MVCNN is presented in this paper[11]. The paper then presents a comparative study of the two methodologies, MVCNN and MVCNN++ (MVCNN using supplementary meta data). Unlike the previous studies which were done on ModelNet dataset, this model is trained on FabWave repository. FabWave repository holds data collected through automated workflow from academic sources and by writing custom web-scrapers for CAD websites. The trained model was then evaluated for its classification and retrieval performance. The newly built MVCNN++ network proved to be 5.8% more efficient than the original MVCNN. This opens up the avenue for more research into the kind of meta data that can be provided to increase the performance of a MVCNN for classification and retrieval purposes.

PointNet, a 3D object classification neural network that takes in point clouds as its inputs is proposed by Charles R. Qi et el in this paper[12]. Unlike other DNN which convert the point cloud into a voxel grid before input, PointNet directly uses these point clouds. This means the model converts the 3D objects into a cloud of points where each point has a value associated with it and then trains the model on these values. Critical point set is defined as the minimum collection of points that is required to efficiently determine the global shape feature for a given 3D object. An upper-bound set the largest possible collection of point clouds that gives the global shape of the 3D object. This model is also trained on the ModelNet dataset. This is one of the simple and effective models to understand. While being effective, the model is robust as well and can handle corruption to the input data to a certain extent[12]. This robustness is due to the fact that a 3D object with set of point clouds between the critical set and upper-bound set gives the same result. Due to this advantage, this model can handle some loss in the data that it is training on.

3D object classification is not just a hot topic in the academic sector, but companies like Google have already rolled out products that employ 3D object classification[13]. Just pointing the lens of your android phone at an object will generate several links for you to explore more about what's in front of your camera. While such products are still in development or in their beta phases, this just shows the potential of 3D object classification and retrieval models beyond the scope of CAD models. Apart from leading tech industries, upcoming websites and communities are also using features like CAD design retrieval via a text-based search. thangs.com is an example of one such website. One can also upload a .stl file to get results similar to the design uploaded.

3. Methodology

This project report explores two of the 3D object classification methods discussed in some of the papers mentioned above, MVCNN and PointNet model. Initially binary classification models of both techniques were prepared for the better understanding of the code and working of the deep neural networks. The images that were used for MVCNN training were JEPG images and STL files were used for modelling of the point clouds from 3D objects. The dataset provided had a total of 46 classes. However, for the training and validations of these models a set of 10 classes was chooses keeping in mind to include some classes with similar shapes and some classes with distinct shapes.

3.1. 2D Binary Classification of Images

For the images to be fed to the model for training, it was important to pre-process these images for more uniformity and standardization in input. The objects to be used in this case did not have any color specific characteristics. Hence, to reduce the processing power required to train the model, all the images are initially converted to grayscale. This drastically reduces the image size and speeds up the preprocessing. To maintain uniformity in the input, all the images are reduced in dimensions and resized to 300 x 300. This also helps to speed up the processing. The image can be reduced to any dimension until it starts to lose its object features. Reducing the images to an extent where object features are lost can lead to degrading of the model performance.

A training dataset was created for the two selected categories – keyway shaft and O-ring. While the two components are significantly different in their shapes, this binary classification was only performed to get a better understanding of the working of convolutional neural networks. The training dataset was the labelled and shuffled to avoid any bias in the training process.

A binary 2D image classification model was built using TensorFlow to understand the working of CNNs and to set a benchmark for comparison with 3D classification of CAD parts. A CNN model was trained on the training dataset. The neural network model had 3 layers one input layer with a convolution layer and a max pooling layer. The second layer also had a convolution layer with a max pooling layer. Next is the hidden layer with 64 neurons. The final layer has a single layer of neuron for the single class output with sigmoid function as the activation function . A sigmoid function can take the value of either 0 or 1[4], hence it is used in case of binary classification.

3.2. 2D Classification of Images (10 classes)

Trial run 1

For classification of 10 classes, all the initial process was the same. The images were gray scaled, resized, labeled, and stored in a training dataset. The dataset was then shuffled to avoid any biases. The only change that was made to the model code before training was that the activation function was changed to ‘softmax’. The softmax function gives out probability outputs for every class. The class with the highest predicted probability is the prediction of the model.

Trial run 2

The initial build model was significantly changed to tackle the problems faced during trial run 1. The convolution matrix size was changed from a 3x3 matrix to 5x5 matrix. This will allow for more granular details to be captured. The number of neurons was increased drastically from 64 to a 1000 in the first hidden layer. Two more hidden layers were added with a 50% dropout rate at the input and 500 and 250 neurons, respectively.

Trial run 3 and Trial run 4

With no significant change in the accuracies of the model from trial run 2, the validation split was increased to 0.3 and 0.4 for trial run 3 and 4 respective to check for any bias in the validation split. This would also help increase the datapoints in validation set.

3.3. 3D Object Classification using PointNet

With little success in building a MVCNN classification model, the next method that was explored was the PointNet model. This model uses point clouds extracted from the 3D object file as input to train the model. This model was initially trained on ModelNet dataset[12] and was then evaluated against ModelNet10 and ModelNet40 classes. Using transfer learning techniques, a new model will be built that will be able to classify the dataset provided. Again, for comparison this model will also be evaluated by classifying 10 classes of different objects.

One of the important variables in using the PointNet model is the number of points used to represent the point cloud of the 3D object. A small number of points can lead to poor training and less accuracy while more than required number of points will result in the the model overtraining on the data or may try to fit on the noise present in the data.

For this model, the training and test datasets were manually created for each individual class of object. A script was written to loop through all the train and test folders of individual classes and then combines into one list that contains the point cloud data for each of the STL file. A different list contained all the labels for these train and test datasets. The next step was to match the cloud point data from train and test dataset to its corresponding match from the label lists. After creating the final train and test dataset with completely labelled parts, the datasets were shuffled individually to avoid any bias in the model training phase.

Trial run 1

For the first trial run, the number of cloud points was selected as 2048. This means every 3D object will have 2048 points representing its shape features in space. The model had 10 classes as its input, and it was run for a total of 5 epochs.

Trial run 2

This run was conducted as a part of continuation of the first run. The only parameter that was changed was the number of epochs. The number of epochs was doubled to 10 runs from the previous 5 epochs. This run was conducted to check if the gradual increase in training and validation accuracies continues and at what time does it start to overfit the data. The number of points per cloud point for every 3D object remained the same at 2048.

Trial Run 3

A third trial was conducted using the same model but the number of points in every point cloud was increased from 2048 to 4096. The model was trained for a total of 6 epochs to keep it comparable to the first trial run with 2048 cloud points. This should enable for the model to pick up on more granular details such as edges and corners of the 3D object. However, this would require more time and more processing power to train the model.

4. Results

4.1. 2D Binary Classification of Images

The model discussed in section 3.1 was trained and then tested on images that were not a part of the training dataset. The model was successfully able to classify the two components. The model details after training show the model accuracy to be 100%. The reason for this perfect accuracy is the significant difference in the shape of the two components tested. The

model accuracy drops when two components of similar shape and external outline are selected. The model details of the binary classification model are represented in figure 3.

Next would be to see how this model performs when multiple classes are introduced, and complexity is increased.

```
Epoch 1/3
18/18 [==============================] - 1055s 59s/step - loss: 2.1217 - accuracy: 0.5666 - val_loss: 0.3185 - val_accuracy:
0.9524
Epoch 2/3
18/18 [==============================] - 880s 49s/step - loss: 0.0668 - accuracy: 0.9982 - val_loss: 0.0031 - val_accuracy:
1.0000
Epoch 3/3
18/18 [==============================] - 882s 49s/step - loss: 3.8431e-04 - accuracy: 1.0000 - val_loss: 5.9158e-05 - val_ac
curacy: 1.0000

<tensorflow.python.keras.callbacks.History at 0x2415b572ca0>
```

Figure 3 : Model details of binary classification model

4.2. 2D Classification of Images (10 classes)

Trial run 1

The multi class model was trained with 5 epochs and a validation split of 0.2. As can be seen from the training results below in figure 4, the model had a very high validation accuracy from the first epoch(almost the double of training accuracy). This could mean either the model was overfitting or there were not enough datapoints present in the validation set. With a believable training accuracy for the first epoch, we can say with some confidence that it's the later and we need to increase the variety and numbers in validation dataset. The model eventually reaches to a training and validation accuracy of 100% which is not ideal. Considering the training data had only 1026 files, the model was also very slow with steps taking as long as 96 seconds in some epoch.

```
model.fit(x, y, batch_size=32, epochs=5, validation_split=0.2)

Epoch 1/5
26/26 [==============================] - 2147s 83s/step - loss: 4.7890 - accuracy: 0.5159 - val_loss: 0.1087 - val_accuracy:
0.9709
Epoch 2/5
26/26 [==============================] - 2507s 96s/step - loss: 0.0599 - accuracy: 0.9768 - val_loss: 0.0086 - val_accuracy:
1.0000
Epoch 3/5
26/26 [==============================] - 1627s 63s/step - loss: 0.0065 - accuracy: 0.9976 - val_loss: 5.5345e-04 - val_accur
acy: 1.0000
Epoch 4/5
26/26 [==============================] - 1471s 57s/step - loss: 9.1276e-04 - accuracy: 1.0000 - val_loss: 0.0028 - val_accur
acy: 1.0000
Epoch 5/5
26/26 [==============================] - 1519s 58s/step - loss: 3.0865e-04 - accuracy: 1.0000 - val_loss: 7.5103e-04 - val_a
ccuracy: 1.0000

]: <tensorflow.python.keras.callbacks.History at 0x138258a4a00>
```

Figure 4: Model details for 2D classification model (10 classes)

Trial run 2

The second trial run was conducted with a new model that had more granularity in its convolutional layer and a greater number of hidden layers with more neurons. This increased the performance of the model with respect to the time taken for training. The time required to train single step went from 96 sec/step at its peak to 7 sec/step. However, there was no change in its training and validation accuracy. Both these values were still not in their ideal limits.

Figure 5 shows the model details for trial run 2.

```
model.fit(x, y, batch_size=32, epochs=5, validation_split=0.2)

Epoch 1/5
26/26 [==============================] - 109s 4s/step - loss: 2.4112 - accuracy: 0.4537 - val_loss: 0.6026 - val_accuracy:
0.8398
Epoch 2/5
26/26 [==============================] - 147s 6s/step - loss: 0.3459 - accuracy: 0.9000 - val_loss: 0.1876 - val_accuracy:
0.9029
Epoch 3/5
26/26 [==============================] - 170s 7s/step - loss: 0.1430 - accuracy: 0.9512 - val_loss: 0.1566 - val_accuracy:
0.9515
Epoch 4/5
26/26 [==============================] - 143s 5s/step - loss: 0.1305 - accuracy: 0.9646 - val_loss: 0.0302 - val_accuracy:
0.9951
Epoch 5/5
26/26 [==============================] - 144s 6s/step - loss: 0.0283 - accuracy: 0.9927 - val_loss: 0.0429 - val_accuracy:
0.9806

]: <tensorflow.python.keras.callbacks.History at 0x2d5b7250b50>
```

Figure 5 : Model details for trial run 2

Trial run 3 and trial run 4

There was no change in the accuracies of the model and both the values stayed very close to 100%
The results can be seen in the figures 6 and 7 below.

```
model.fit(x, y, batch_size=32, epochs=5, validation_split=0.3)

Epoch 1/5
23/23 [==============================] - 118s 5s/step - loss: 0.0393 - accuracy: 0.9889 - val_loss: 0.0096 - val_accuracy:
0.9968
Epoch 2/5
23/23 [==============================] - 130s 6s/step - loss: 0.0271 - accuracy: 0.9916 - val_loss: 0.0176 - val_accuracy:
0.9968
Epoch 3/5
23/23 [==============================] - 129s 6s/step - loss: 0.0161 - accuracy: 0.9986 - val_loss: 0.0037 - val_accuracy:
0.9968
Epoch 4/5
23/23 [==============================] - 138s 6s/step - loss: 0.0041 - accuracy: 0.9986 - val_loss: 0.0014 - val_accuracy:
1.0000
Epoch 5/5
23/23 [==============================] - 130s 6s/step - loss: 0.0040 - accuracy: 0.9972 - val_loss: 0.0029 - val_accuracy:
0.9968

5]: <tensorflow.python.keras.callbacks.History at 0x2d5b76a7850>
```

Figure 6: Model details for trial run 3

```
model.fit(x, y, batch_size=32, epochs=5, validation_split=0.4)

Epoch 1/5
20/20 [==============================] - 109s 5s/step - loss: 0.0045 - accuracy: 0.9984 - val_loss: 8.7680e-04 - val_accurac
y: 1.0000
Epoch 2/5
20/20 [==============================] - 123s 6s/step - loss: 7.2193e-04 - accuracy: 1.0000 - val_loss: 7.9978e-04 - val_acc
uracy: 1.0000
Epoch 3/5
20/20 [==============================] - 128s 6s/step - loss: 0.0041 - accuracy: 0.9984 - val_loss: 0.0115 - val_accuracy:
0.9976
Epoch 4/5
20/20 [==============================] - 129s 6s/step - loss: 0.0049 - accuracy: 0.9984 - val_loss: 0.0021 - val_accuracy:
0.9976
Epoch 5/5
20/20 [==============================] - 131s 7s/step - loss: 0.0133 - accuracy: 0.9951 - val_loss: 0.0056 - val_accuracy:
0.9976

]: <tensorflow.python.keras.callbacks.History at 0x2d5b524f550>
```

Figure 7: Model details for trial run 4

4.3. 3D Object Classification using PointNet

Trial run 1

The model was run for 5 epochs and seemed to be working as intended. The training as well as validation accuracy was seen gradually increasing with every epoch. The training accuracy jumped from 33.67% at the first epoch to 72.28% for the final epoch. The validation accuracy also showed a decent increase from 23.53% to 55.88%. However, with accuracies still increasing the model stopped training because it reached the pre-defined epochs. It is possible that the same model can give better accuracies if trained for more epochs. This has been explored in detail in the next trail run.

The model detail and the prediction outputs for the first trial run can be seen below in figure 8 and figure 9.

```
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["sparse_categorical_accuracy"],
)

model.fit(train_dataset, epochs=5, validation_data=test_dataset)
```

```
Epoch 1/5
31/31 [==============================] - 155s 5s/step - loss: 3.6371 - sparse_categorical_accuracy: 0.3367 - val_loss: 3.239
9 - val_sparse_categorical_accuracy: 0.2353
Epoch 2/5
31/31 [==============================] - 165s 5s/step - loss: 2.6956 - sparse_categorical_accuracy: 0.5131 - val_loss: 21.26
43 - val_sparse_categorical_accuracy: 0.3235
Epoch 3/5
31/31 [==============================] - 164s 5s/step - loss: 2.5198 - sparse_categorical_accuracy: 0.5655 - val_loss: 5.079
4 - val_sparse_categorical_accuracy: 0.4706
Epoch 4/5
31/31 [==============================] - 165s 5s/step - loss: 2.2072 - sparse_categorical_accuracy: 0.6452 - val_loss: 4.018
0 - val_sparse_categorical_accuracy: 0.5882
Epoch 5/5
31/31 [==============================] - 165s 5s/step - loss: 2.0508 - sparse_categorical_accuracy: 0.7228 - val_loss: 2.104
1 - val_sparse_categorical_accuracy: 0.5588

<tensorflow.python.keras.callbacks.History at 0x21b77c7d730>
```

Figure 8 : Model details for PointNet model – Trial Run 1

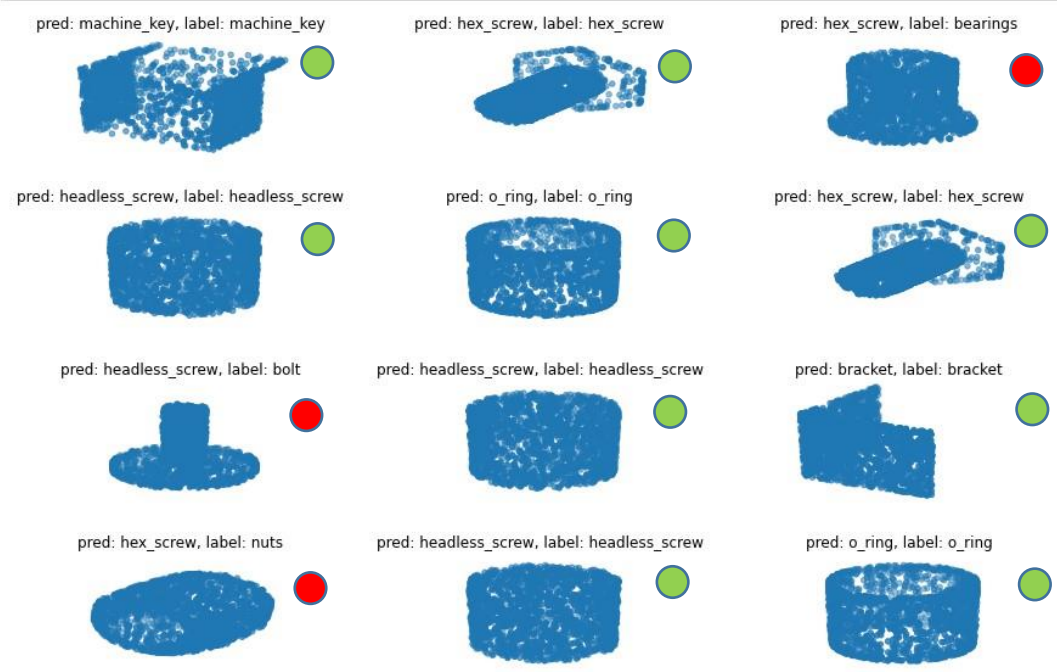

Figure 9: Prediction results for PointNet model Trial run 1

Trial run 2

The second trial run had better result and accuracies compared to the first trial run. This run was trained on 10 epochs. The training accuracy for this model is 89.72% which is an almost 18% rise from the first trial run. The validation accuracy also increased to 79.41% from 55.88% of the initial run. However, the validation accuracy had peaked at 91% for some epochs, which was even more than the training accuracy for that epoch. This can be attributed to some random bias being introduced to the validation dataset for these particular epochs. A little more data in the validation dataset and making sure there is no human bias induced in the selection of validation data can solve this issue. Given the results of first trial run and the second trial run, this little random bias in 3 particular epochs is not a cause of great concern and can be solved with the ways that will be discussed later in this report.
The details for the model and the prediction outcomes are presented below in figure 10 and figure 11, respectively.

```
In [61]:
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["sparse_categorical_accuracy"],
)

model.fit(train_dataset, epochs=10, validation_data=test_dataset)
```

```
Epoch 1/10
31/31 [==============================] - 234s 8s/step - loss: 3.6439 - sparse_categorical_accuracy: 0.3679 - val_loss: 3.187
7 - val_sparse_categorical_accuracy: 0.2353
Epoch 2/10
31/31 [==============================] - 196s 6s/step - loss: 2.7181 - sparse_categorical_accuracy: 0.5030 - val_loss: 2.863
2 - val_sparse_categorical_accuracy: 0.2353
Epoch 3/10
31/31 [==============================] - 198s 6s/step - loss: 2.5072 - sparse_categorical_accuracy: 0.5645 - val_loss: 3.116
2 - val_sparse_categorical_accuracy: 0.5882
Epoch 4/10
31/31 [==============================] - 196s 6s/step - loss: 2.2001 - sparse_categorical_accuracy: 0.6573 - val_loss: 1.961
1 - val_sparse_categorical_accuracy: 0.7941
Epoch 5/10
31/31 [==============================] - 193s 6s/step - loss: 1.9878 - sparse_categorical_accuracy: 0.7399 - val_loss: 2.161
7 - val_sparse_categorical_accuracy: 0.7941
Epoch 6/10
31/31 [==============================] - 273s 9s/step - loss: 1.7974 - sparse_categorical_accuracy: 0.8145 - val_loss: 2.398
6 - val_sparse_categorical_accuracy: 0.6765
Epoch 7/10
31/31 [==============================] - 281s 9s/step - loss: 1.6306 - sparse_categorical_accuracy: 0.8619 - val_loss: 1.510
4 - val_sparse_categorical_accuracy: 0.9118
Epoch 8/10
31/31 [==============================] - 278s 9s/step - loss: 1.5733 - sparse_categorical_accuracy: 0.8831 - val_loss: 1.413
0 - val_sparse_categorical_accuracy: 0.9118
Epoch 9/10
31/31 [==============================] - 293s 9s/step - loss: 1.5701 - sparse_categorical_accuracy: 0.8740 - val_loss: 1.702
2 - val_sparse_categorical_accuracy: 0.9118
Epoch 10/10
31/31 [==============================] - 273s 9s/step - loss: 1.4941 - sparse_categorical_accuracy: 0.8972 - val_loss: 1.844
9 - val_sparse_categorical_accuracy: 0.7941
```

```
Out[61]: <tensorflow.python.keras.callbacks.History at 0x21b0c0c49a0>
```

Figure 10: Model details for PointNet model trail run 2

Trial run 3

This run was carried out with 4096 points in every point cloud of individual 3D object and was run for 6 epochs. The model showed decent performance and was very comparable to the model trained with 2048 points. The last epoch again showed higher accuracy for validation dataset than training dataset. This can be due to overfitting on the validation dataset. Omitting the values for that particular epoch, we can say that the model had trainings accuracy of 55.75% and validation accuracy of 47.06% from the previous run. As this model did not outperform the previous model using 2048 points, keeping in mind the tradeoff for processing time and power and the results of this model, it was decided to not pursue this model further. The model with 2048 cloud points performs in a more accurate and fast way. The addition of extra 2048 cloud points did not have any significant advantage.
The model details and the prediction outcomes are mentioned below in figure 12 and figure 13, respectively.

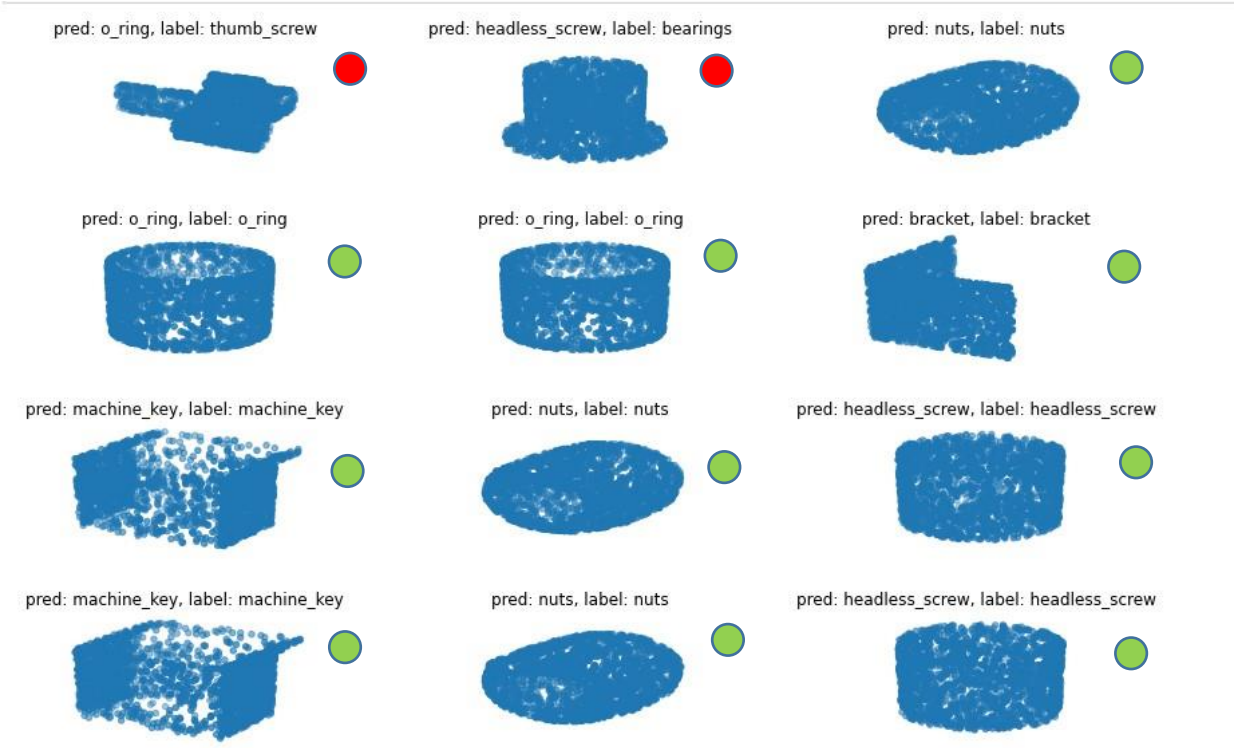

Figure 11: Prediction results for PointNet model Trial run 2

```
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["sparse_categorical_accuracy"],
)

model.fit(train_dataset, epochs=6, validation_data=test_dataset)
```

```
Epoch 1/6
31/31 [==============================] - 329s 11s/step - loss: 3.6642 - sparse_categorical_accuracy: 0.3891 - val_loss: 3.37
48 - val_sparse_categorical_accuracy: 0.2353
Epoch 2/6
31/31 [==============================] - 338s 11s/step - loss: 2.8433 - sparse_categorical_accuracy: 0.4889 - val_loss: 3.00
65 - val_sparse_categorical_accuracy: 0.2353
Epoch 3/6
31/31 [==============================] - 339s 11s/step - loss: 2.6228 - sparse_categorical_accuracy: 0.5202 - val_loss: 3.08
17 - val_sparse_categorical_accuracy: 0.2353
Epoch 4/6
31/31 [==============================] - 340s 11s/step - loss: 2.5859 - sparse_categorical_accuracy: 0.5131 - val_loss: 5229
61.6562 - val_sparse_categorical_accuracy: 0.3529
Epoch 5/6
31/31 [==============================] - 420s 14s/step - loss: 2.4985 - sparse_categorical_accuracy: 0.5575 - val_loss: 2.64
60 - val_sparse_categorical_accuracy: 0.4706
Epoch 6/6
31/31 [==============================] - 594s 19s/step - loss: 2.2635 - sparse_categorical_accuracy: 0.6381 - val_loss: 2.34
44 - val_sparse_categorical_accuracy: 0.7059
```

```
]: <tensorflow.python.keras.callbacks.History at 0x21b09f640a0>
```

Figure 12: Model details for PointNet model trail run 3

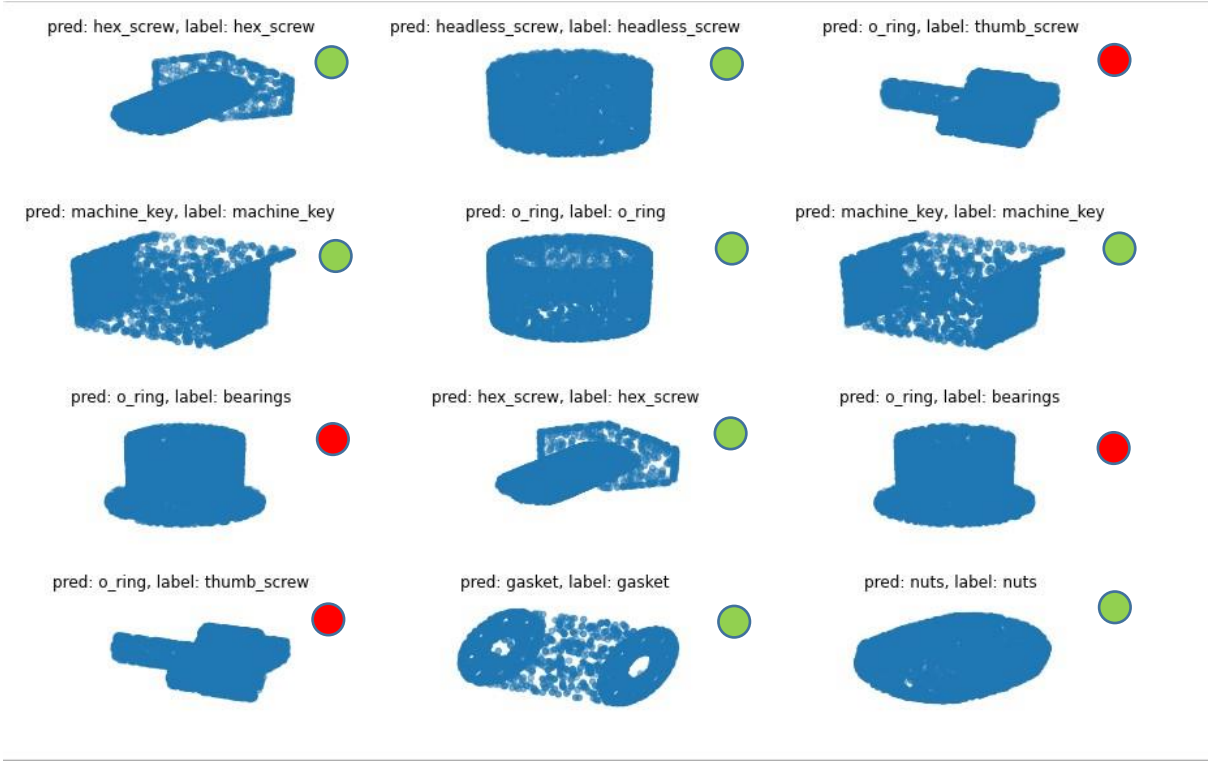

Figure 13: Prediction results for PointNet model Trial run 3

5. Discussion

In this section the methodologies and the results of the two models developed above are discussed.

5.1. 2D Image Classification Model

A binary image classification model was developed to check for the feasibility of building a new model to classify 3D objects using 2D images. The model built was a CNN model. The binary classification model was tested on two very distinct components, keyway shaft and 0-ring. Both of these objects are very dissimilar and very easy to classify. The model performed very well on these classes as expected.

The model was then updated to handle multi-class classification. The input provided to this model was multiple images of a single object form different views. This model tries to replicate the working of a MVCNN model. The model was evaluated on 10 classes of different objects. The model did not perform very well and ended up overfitting the data with an accuracy of 100%. The model also required high processing power and time to train. It was concluded that the model built was not powerful enough to process the input data. To overcome this issue, the neural network model was drastically changed to include more layers and more neurons to help increase the processing speed and capacity. The convolutional layers in this new model were also changed into 5x5 matrix for more granularity in the hopes that it will increase the efficiency. After testing, it was found that this did not help in improving the model performance in terms of accuracy, but this did improve the model speed.

To solve the issue of validation accuracy being too high and sometimes even higher than the training accuracy multiple test runs were performed with increasing the size of validation dataset. One reason for validation accuracy to be higher than training accuracy is that the validation dataset contains too few datapoints or has an imbalance of number of datapoints from each category[14]. After multiple runs with increasing the size of validation dataset the model still showed the same results.

In future for running a MVCNN model more variations can be tested in case of model design to find an optimal solution. Instead of building a new model, techniques like transfer learning can also be used to train models in a quick and efficient way. Transfer learning uses an already available model to train and customize your own model to perform a specific task[15]. Transfer learning technique was used build the 3D classification model built in this project study. More about this is discussed in a later section.

The image data can also be supplemented with other metadata that is generated by a CAD software. This meta data can include dimensions, shape and size details, orientation detail or any other supportive data. This method has previously shown to improve the performance of MVCNN model.[11]

This study focused on a limited amount of data. The study was performed only on 10 classes with a total training dataset length of 1026 images. Increasing the size of data can drastically improve the performance of this model. Also, the image data that was used lacked 'multi view' images and that could have led to a certain degradation in the performance of the model. The sample of image data that was used is provided in figure 14. The data lacks multi-view aspect of the 3D object. By using multiple cameras and using high resolution photography a small number of multi view images can be tested as an input to this model to check for its performance.

Figure 14: Sample of the image data used for training

5.2. 3D Object Classification using PointNet

The 3D object classification model that is built in this project study uses the PointNet model to learn from using the transfer learning technique. The PointNet model intakes the point cloud of a 3D object as the input and uses this as the data to train the model. In this study the same 10 classes used for image classification are used to evaluate the performance of this 3D object classification model trained from PointNet.

This model is heavily inspired by the PointNet model developed in the study published by Stanford University[12]. The input used for this model is STL file format. The total number of datapoints available for training are 992 STL files. For the first trial run the number of points taken into consideration from each 3D object file is 2048. This number of points is an important parameter that affects the performance of the model. The first trial run yielded decent results with 55.88% validation accuracy after just 5 epochs. This prompted to run a second trial with a greater number of epochs. The second trial was performed for 10 epochs and showed improved results with 89.72% training accuracy and 79.41% validation accuracy.

To check for even better performance, a third trial run was initiated that used the same model but doubled the cloud points for every 3D object used. This model did not have very impressive results as the training and validation accuracy was not better than trial run 1 which used half the number of cloud points. This observation leads to the conclusion that there is no incentive in pursuing this model further to check for model improvements. This supports the finding that point cloud that falls between critical point set and upper-bound point set gives the same result[12]. Critical point set is defined as the minimum number of cloud points that will be required to determine the global shape features. The upper-bound shape is the point cloud that represents every possible point in the 3D object.

An interesting find regarding this model is all the three models predicted thumb screw as O-ring. Both these classes are very dissimilar from each other, yet all the three models categorized a thumb screw as an O-ring. This can be attributed to the fact that the dataset had only 25 training points available for a thumb screw and 372 datapoints for an O-ring. This disparity in numbers of individual category datapoints can also lead to decreasing performance. Considering the data disparity in some of the categories, this model performs satisfactorily with the given accuracies.

Table 1 shown below compares the performance of all three trial runs for their accuracies.

Trial Run	Number of cloud points	Number of epochs	Training accuracy	Validation accuracy
1	2048	5	72.28%	55.88%
2	2048	10	89.72%	79.41%
3	4096	5	55.75%	47.06

Table 1: Model comparison for 3D object classification model

The model discussed in the paper only classifies 10 classes. This can be improved upon by adding in more classes for classification as future work. Training this model on larger dataset using machines with higher computing power can be done to increase the scope of this model from just manufacturing components to more general use categories like kitchen or office.

This study considered image data and 3D data for classification of 3D objects. The studies scope was limited to classification of 3D objects. Further research and work can be put in to develop a model that is also capable of 3D object retrieval from a large dataset. The scope of use for 3D object classification and retrieval model is immense in the field of automated manufacturing, advanced manufacturing of healthcare equipment, retrieval of CAD models from large public repositories. It might also be worth looking into models that train on both multi-view image data and 3D object files like STL. Using images to classify 3D objects and using 3D files to classify 3D objects, both have their advantage and disadvantages. It can be interesting to see the model performance if both methods are used as inputs to a single model.

6. Conclusion

Advanced neural networks are very powerful mechanism and have been used for a variety of different applications like predictive analytics, forecasting, computer vision application, autonomous driving. This paper explores two of the popular type of neural network algorithms that have been used in classification and retrieval of 3D objects.

This project study failed to design a unique MVCNN algorithm that was capable of classifying the provided dataset. This could have been a result of the lack of variation in 'multi-view' aspect of the selected data. Testing this data on a different dataset could help clear up this speculation. The model could have also been built using transfer learning techniques but the second model using 3D objects as inputs was built using transfer learning, so this opportunity was used to try to build a unique model that also provided a huge learning opportunity. A larger dataset could have also helped and prevented the model from overfitting.

This study was successful at building a custom 3D object classification model for the provided data using the transfer learning technique. The model used for transfer learning was PointNet. This study also conducted 3 trial runs using this model. The trial runs helped to understand the effect of number of points in a point cloud used as the input for a 3D object. For the dataset used in this study, it was found that 2048 points for every 3D object was the optimal number allowing for quick and accurate model training and prediction. The model using 4096 data points did not improve significantly on the performance of previous model and was not worth the extra time, memory, and processing power. However, it is important to note that the optimal number of cloud points was decided for this particular dataset used in this study.

Acknowledgements

I would like to acknowledge the support of Prof. Dr. Binil Starly for his support throughout the project and for providing access to the dataset used in this project.

All the code that has been used for this project study is available on the GitHub link provided below.
https://github.ncsu.edu/pmujumd2/ISE_789_Project/tree/main

References

[1]https://www.altexsoft.com/blog/datascience/machine-learning-use-cases-in-finance/

[2]https://www.businessinsider.com/artificial-intelligence-healthcare

[3]https://www.logisticsit.com/articles/2019/05/15/research-finds-manufacturing-industry-hindered-by-a-lack-of-data-skills,-but-businesses-are-working-hard-to-address-this/

[4] https://www.geeksforgeeks.org/activation-functions-neural-networks/

[5]https://www.ibm.com/cloud/learn/neural-networks#:~:text=Neural%20networks%2C%20also%20known%20as,neurons%20signal%20to%20one%20another.

[6]https://www.researchgate.net/figure/Neural-network-architecture-Figure-3-presents-a-neural-network-active-node-25-with_fig2_333004520

[7] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository

[8] Zhirong Wuy, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes

[9] Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, Dec. 11–18, pp. 945–953.

[10] Qian Yu, Chengzhuan Yang, Honghui Fan, Hui Wei. Latent-MVCNN: 3D Shape Recognition UsingMultiple Viewsfrom Pre-defined or Random Viewpoints. Neural Processing Letters (2020) 52:581–602

[11] Atin Angrish, Akshay Bharadwaj, Binil Starly. MVCNN++: Computer-Aided Design Model Shape Classification and Retrieval Using Multi-View Convolutional Neural Networks. Journal of Computing and Information Science in Engineering. February 2021, Vol. 21

[12] Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. Proceedings of the IEEE International Conference on Computer Vision

[13]https://towardsdatascience.com/google-objectron-a-giant-leap-for-the-3d-object-detection-9d8393b7a183

[14] https://stackoverflow.com/questions/42878683/keras-cifar10-example-validation-and-test-loss-lower-than-training-loss/42924616#42924616

[15] https://www.tensorflow.org/tutorials/images/transfer_learning