Research Project

Utilizing Artificial Intelligence (AI) For 3D Models

Author: Yohanes Sugiarto
Politeknik ATMI Surakarta

Ana Ningsih
Politeknik ATMI Surakarta

1 Table of Contents

2 Summary

Almost every industry that uses computers and digitalization is now expanding its technology implementation toward Artificial Intelligence (AI). The more activities that use information and communication technology, the more likely it is that AI can be integrated into these activities.

In computer studies, the topic of artificial intelligence usually narrowed down to more specific terms that refer to the methods by which the intelligence is achieved, namely Machine Learning (ML) and Deep Learning. Both Machine Learning and Deep Learning need to learn from the observed data first, to then generate a solution or algorithm for a specific problem.

In the standard Machine Learning approaches, the unique features of the input data need to be identified first, before the data is entered into the models. Deep Learning does not need such features extraction. The features that specify a solution will be directly learned from the data. Therefore, Deep Learning requires more data than standard Machine Learning but less engineer's intervention during the process.

In recent years, Deep Learning has demonstrated remarkable capabilities in solving 2D graphic tasks such as image classification, object detection, semantic segmentation, etc. Currently, Deep Learning has also shown tremendous progress in solving 3D graphic tasks.

This project examines the implementation of Deep Learning for the classification and generation/reconstruction of 3D objects. Using the Python programming language, two Deep Learning frameworks, TensorFlow and PyTorch, are utilized to create artificial neural networks. The results are impressive. They give so much promise for further development and implementation. Using a relatively simple Deep Learning model with only a few learning iteration, most of the neural networks models in this project are able to yield an average accuracy rate of over 60%.

From the internal side, the capability of existing AI models can still be improved by optimizing and fine-tuning the hyper parameters or by incorporating existing state-of-the-art networks. From the external side, there are still so many areas that have not taken advantage of AI yet.

3 Introduction

3.1 Project Description

Currently, almost all sectors that use computers and digitalization are expanding their technology implementations toward Artificial Intelligence (AI). Like information and communication technology, AI can be integrated into various fields. The more activities utilize information and communication technology, the more likely AI can be integrated into those activities. Many AI solutions have been incorporated into systems in such a way that sometimes we as users are not aware of their existence.

In the past, many considered that AI was closely related to robotics because it was generally known that its emergence began with the development of robotics technology. Today, AI is becoming ubiquitous, and it is not just related to robotics. AI offers new solutions to solve complex problems. In the health sector, AI can be used to detect diseases with the help of expert systems. In the restaurant business, robots can be applied as substitutes for waiters. In the warehouse management, intelligence lines can be applied to sort and deliver goods. In the game development, state-of-the-art non-player characters (NPC) are becoming as intelligent as the players themselves. In transportation, self-driving cars have become smarter. And so on. Basically, AI can be found in almost every place.

In computer studies, the topic of artificial intelligence usually narrows to a more specific term that refers to the method by which the intelligence is achieved, namely Machine Learning (ML). The term of Machine Learning implies that a machine or computer needs to learn the data first, to come up with a solution or an algorithm for a problem. Usually, for a particular problem, there will be a specific model of Machine Learning. Machine Learning models are oriented toward one specific task to make accurate predictions. An ML engineer will highlight specific features to be learned by the machine to identify the correct solution.

A further term that is getting more popular is Deep Learning. Deep Learning can be considered as a sub-discipline of Machine Learning. Deep Learning requires much more data than normal Machine Learning but less engineer's intervention during the process. The features that specify a solution will be directly learned from the data. Deep learning is able to make more accurate predictions, but it will take a longer time to train or learn.

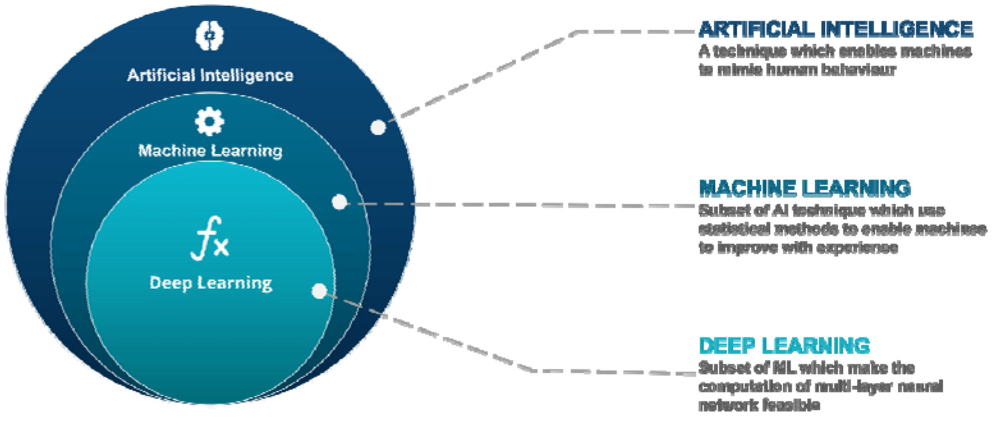

Artificial Intelligence
Machine Learning
f_x
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 3-1. AI, ML, and DL (Babu, 2019)

The increase of computer performance boosts the popularity of deep learning. There are many different fields in which deep learning is applied, such as:

1. Natural Language Processing (NLP)

   Applications in this field for example: auto-completion, speech recognition, language translation, etc.

2. Computer Vision

   Applications in this field for example: object detection, face detection, self-driving car, etc.

3. Image and audio synthesis

   Applications in this field for example: image restoration, scenes generation in games, voice generation, music composer, etc.

4. Signal processing

   Applications in this field for example: noise detection, denoising signals, etc.

Generally, AI opens many possibilities that have never existed before. There is tremendous progress in fields that usually easy for human but difficult for machines.

3.2 Goals

The main objective of this project is to study the applications of artificial intelligence in computer vision and modeling, both for 2D and 3D. The focus of the project is the implementation of deep learning using existing frameworks.

The following tasks provide an overview of the steps, studies, and areas to be explored in order to fulfill the main objective:

- Include reviews of related works and concepts that previously have been studied and developed
- Hands-on experience with existing models and methods that use deep learning frameworks
  - Data exploration
  - Exploration on model selection
  - Exploration on hyper-parameter tuning
  - Performance measures
- Applying the hands-on into the selected 3D model realm

3.3 Motivation

The inspiration and motivation for this project came from the hype and widespread use of AI in almost all aspects of life. Technological advancements are increasing rapidly, huge performance improvements are happening on daily basis. This allows the use of AI quite easily. The term AI is becoming a household term and very popular.

In the field of computer vision, AI is widely used, starting from simple image recognition to complex obstacle and collision detection for automated or self-driving vehicles, and even the creation of luxury arts.

The need to connect the dots between 2D vision to 3D model is also increasing. The use of AI in the field of 3D design and modeling is growing. In line with my current activity in 3D modeling and programming, it is important to study and examine the possibility of widening the current developments by applying AI solutions.

4 Literature Review

4.1 Machine Learning

There are many definitions of Machine Learning. This field is advancing rapidly, and more sub-disciplines are continually progressing. In general, Machine Learning consists of computer methods which analyze observation data to automatically detect patterns and use the uncovered patterns to perform tasks based on new unobserved data. This definition was stated by K. Murphy in 2012.

The first definition of Machine Learning was given by Arthur Samuel in 1959: "Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed." A formally definition was also stated by Tom Mitchell from Carnegie Mellon University in 1997: "A Machine Learning program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with Experience E."

The performance measure stated in the latter definition is a key concept to get the mathematical equations for the learning process. A Machine Learning model learns existing data and then create some sort of structure based on the data. This model will be then the agent for solving future tasks on new data. This procedure can be pictured as following diagram:

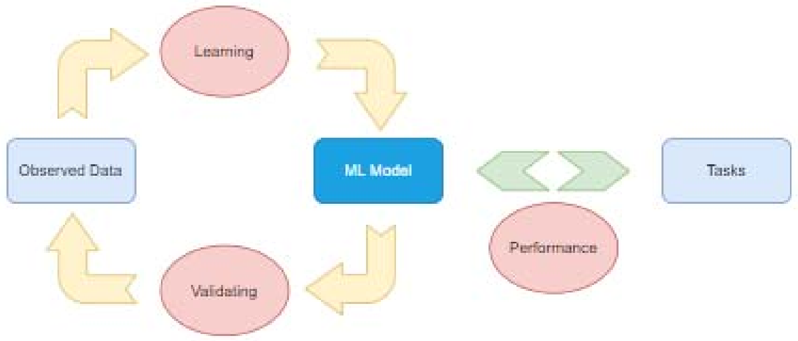


Figure 4-1. Machine Learning Concept

4.1.1 Type of Machine Learning

There are three types of Machine Learning, namely:

- **Supervised Learning**

  In supervised learning, the machine is trained on a labeled dataset. It contains the input data and the resulting output. The machine has to find its own algorithm, so that it can predict an accurate output when given new data.

  Supervise learning is usually used for classification and regression problems.

- **Unsupervised Learning**

  In unsupervised learning, the machine is trained on a dataset which does not have any labels nor any information about the output, whatsoever. The machine must find similarities, patterns, and relationships inside the data. It has no specific output prediction, but it can give categorizations.

  Unsupervised learning is usually used for clustering problems.

- **Reinforcement Learning**

  In reinforcement learning, the machine is trained using reward and punishment scenario. The machine use trial and error to determine which methods deliver the most rewards. The more accurate the result, the machine will get a reward. On the other hand, if the result tends to worsen, the machine will get a punishment. This type of learning is suitable for optimization problems.

4.1.2 Machine Learning Process

Machine Learning process as overviewed in Figure 4-1 can be defined as sequence of following steps:

- **Data collection**

  This is where data is gathered from various sources. Data can be in many different formats.

- **Preprocessing**

  This is where data is analyzed and filtered to meet the requirements of the Machine Learning. The data will also be split into training set and test set. Training set will be used for training, test set will be used for validation.

  In this step, feature extraction can also be done to reduce the data into set of relevant features.

- **Training**

  This is where the machine is learning. Its algorithm will be trained using training set.

- **Evaluation**

  This is where the algorithm is evaluated using the test set.

- **Optimization**

  This is where the machine adjusts its algorithm and parameters to increase accuracy.

- **Postprocessing**

  Postprocessing is necessary if the output of the machine need to be adjusted to match the output criteria, e.g., adjusting class scores into class decision.

4.2 Deep Learning

Deep Learning is a sub-branch of Machine Learning. Deep Learning is inspired by how the brain works. A brain consists of billions of interconnected neuron cells. Deep Learning imitates this using interconnected artificial neurons. This is where the term Artificial Neural Network (ANN) comes. The connections between neurons determine how the output will be. Usually, the larger the data, the more number of neurons is needed.

Deep Learning differs from Machine Learning in several ways. First, Deep Learning requires a significantly larger dataset. Second, which is quite a principle, Deep Learning does not require features engineering, it will extract all features during its learning process. Deep Learning is actually more flexible and powerful, but it requires longer learning process.

4.2.1 Artificial Neural Network (ANN)

Basically, an Artificial Neural Network consists of three layer, as shown in the following image.

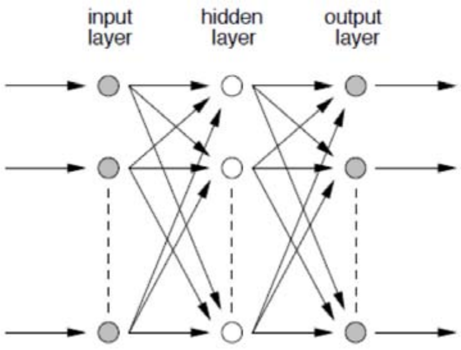


Figure 4-2. Shallow Neural Network

- **Input Layer**

  The input layer contains neurons which receive the input data. Those neurons then pass the information to the hidden layers. This layer does not perform any computation.

- **Hidden Layer**

  The hidden layer can contains multiplle hidden layers. Figure 4-2 has only one hidden layer. This layer perform the computation and pass the result to the output layer.

- **Output Layer**

  This is the final layer which perform the final computation to get the final result.

4.2.2 Computation in ANN

The computation in Artificial Neural Network is quite straight forward. Connections between neurons have numerical values which called weights. All weights connected to a neurons will be summed. The result will be fed to the activation function (step function, in Figure 4-3).

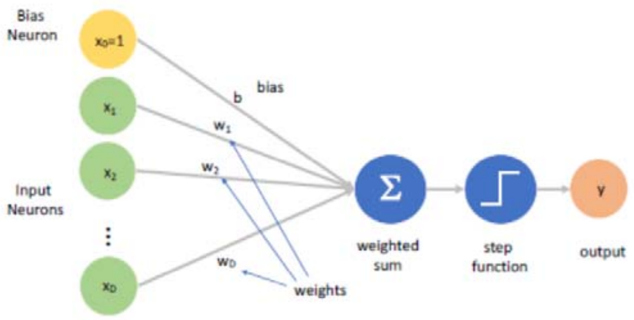


Figure 4-3. Cell Computation

4.3 Machine Learning Frameworks

Python is currently the most widely used programming language in Data Science and Machine Learning. Python is full of features and a platform for doing research. One of the important things in Python is Machine Learning frameworks. There are many Machine Learning libraries or frameworks that can be used in Python. The most wellknown Machine Learning frameworks are TensorFlow and PyTorch. Both are widely used in Deep Learning. Both are free and open-source.

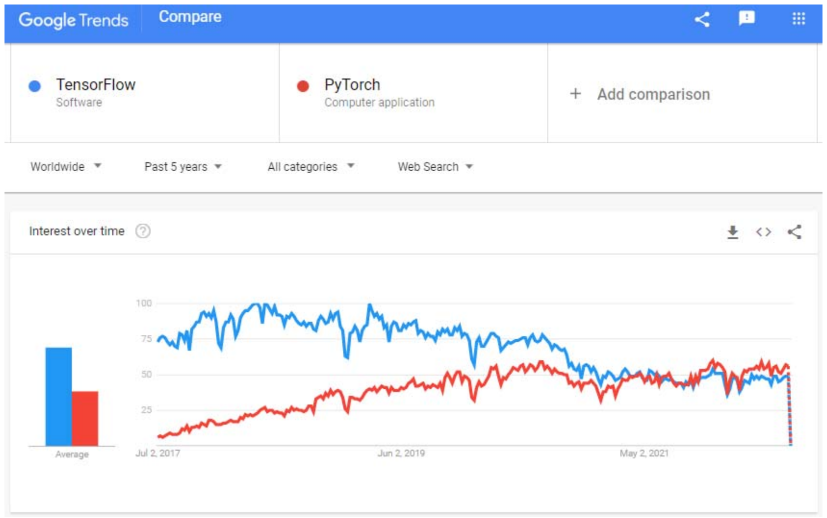


Figure 4-4. TensorFlow vs PyTorch

According to GoogleTrends[1], both are neck and neck in popularity. TensorFlow started earlier, but its popularity declined over time and now PyTorch is already on the slightly higher level.

4.3.1 TensorFlow

TensorFlow[2] is free and open-source deep learning framework developed by Google based on Theano. Tensorflow is known as an industry-focused framework. TensorFlow is widely used by companies and businesses and now it gains more popularity in the research community.

TensorFlow

Figure 4-5. TensorFlow Logo

Several key aspects of TensorFlow:

- This framework is matured and exists for a long time.
- Popular among industry professionals.
- Support deployment into production.
- Accompanied by visualization tool of Tensorboard.

[1] https://trends.google.com/
[2] https://www.tensorflow.org/

- Faster and high performance learning.

TensorFlow was initially based on a static graph execution. Static graph requires that the Machine Learning model composition is defined at the beginning, there is no possibility to tweak or tune the model during the learning process. However, the newer version allows for eager mode like PyTorch and allows for real Phyton programming style.

Keras

Keras is a high-level Application Programming Interface (API) that facilitates Deep Learning programming. It is an free and open-source library. Keras is integrated in the TensorFlow framework. Keras API is user friendly, easy to use, easy to understand. In Keras, neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that can be combine to create new Machine Learning models.

4.3.2 PyTorch

PyTorch[3] is free and open-source deep learning framework developed by Facebook based on Torch. Although it came one year later after TensorFlow, PyTorch is very popular among researcher. PyTorch is known as a research-focused framework. It is also known for its simplicity, ease of use, and efficient memory usage.

Figure 4-6. PyTorch Logo

Several key aspects of PyTorch:

- This framework is relatively new, but increasingly popular.
- Popular among researchers.
- No support deployment into production, third-party applications needed.
- No visualization tool, third-party tools needed.
- High performance learning but relatively slower than TensorFlow.

PyTorch is based on dynamic eager execution. The components of Machine Learning model is constructed dynamically.

4.4 Additional Libraries

4.4.1 PyTorch3D

PyTorch3D[4] is a modular and optimized library specifically for 3D Deep Learning using PyTorch. It is also developed by Facebook. Working with 3D data is quite challenging. PyTorch3D supposed to accelerate those kind of work.

[3] https://pytorch.org/

[4] https://pytorch3d.org/

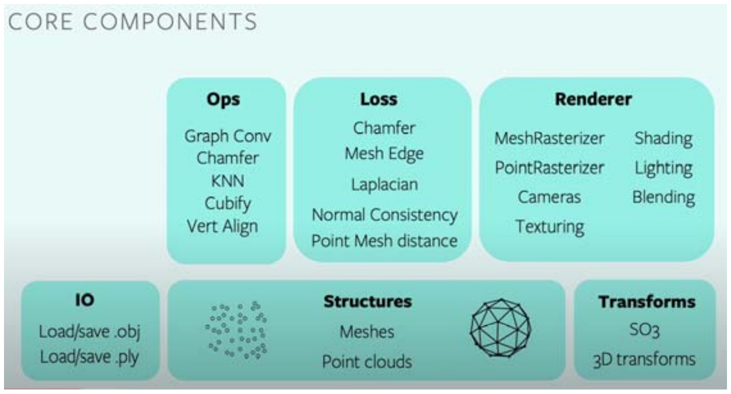


Figure 4-7. PyTorch3D Core Components

PyTorch3D has 3D operators, heterogeneous batching capabilities, and a modular differentiable rendering API. Key features include: data structure for storing and manipulating triangle meshes; efficient operations on triangle meshes (projective transformations, graph convolution, sampling, loss functions); and a differentiable mesh renderer.

4.4.2 Kaolin

Kaolin is also a PyTorch library for 3D Deep Learning. It is developed by NVIDIA. Kaolin provides an API for working with a variety of 3D representations and includes a growing collection of GPU-optimized operations such as modular differentiable rendering, fast conversions between representations, data loading, 3D checkpoints and more.

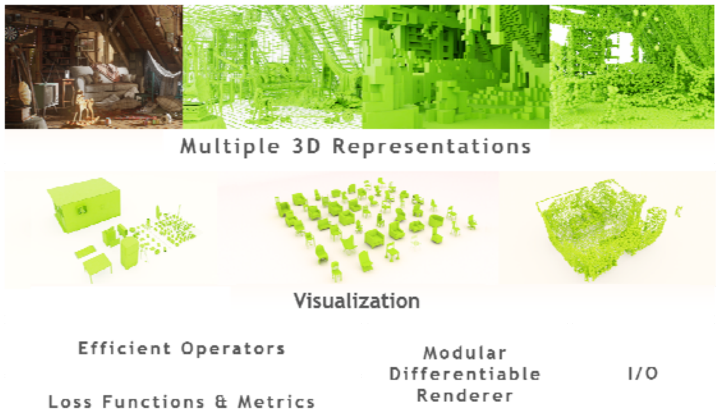


Figure 4-8. Kaolin Core Components

4.5 Paper Study

4.5.1 Survey and Evaluation of Neural 3D Shape Classification Approaches

This paper takes the topic of classification of 3D objects. The main focus is supervised learning, specifically the classification task, which is closely related to global feature extraction.

The researchers conduct an extensive survey of existing Deep Learning based 3D shape classification approaches and categorize them based on the common approach ideas. They also evaluate 11 selected classification networks on three 3D object datasets, extending the evaluation to a larger dataset on which most of the selected approaches have not been tested yet.

The categorization of the networks is based on the shape of the input, namely volumetric grid-based, multiple-viewpoint image-based, point cloud-based, networks which process the object's shape or mesh approximation, and hybrid methods that process multiple representations simultaneously. Basic representation types are illustrated in Figure 4-9.

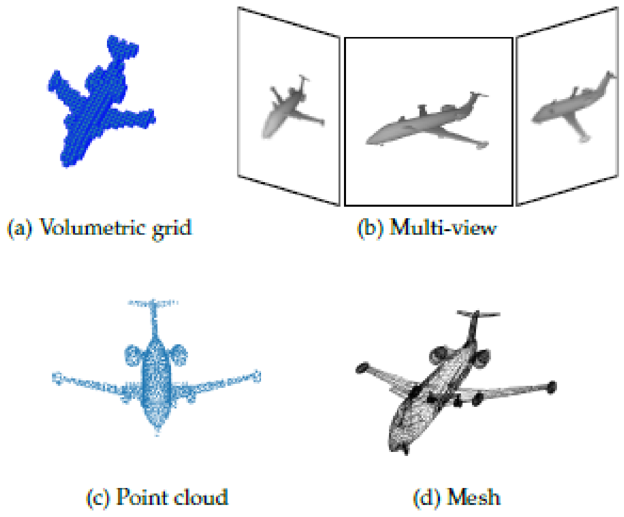
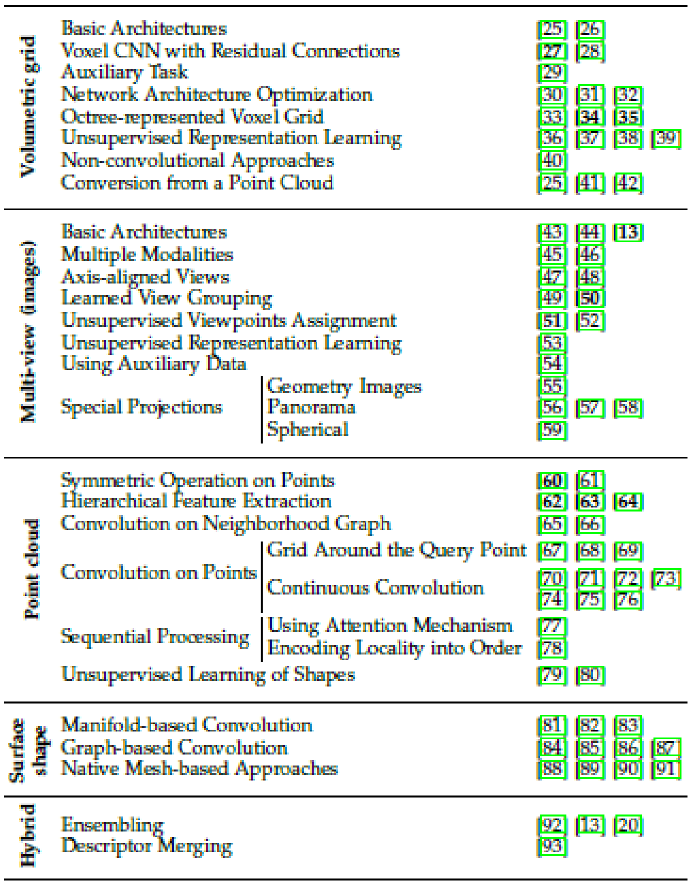


Figure 4-9. 3D Representations for NN Input

Classification neural network architectures can be divided into two parts: a **feature extractor**, which transforms the input shape representation to a feature vector, called descriptor, and a **classifier**, which learns to transform the extracted features into scores denoting the probability of individual classes. The surveyed approaches are shown in the following picture:



Volumetric grid	Basic Architectures		[25] [26]
	Voxel CNN with Residual Connections		[27] [28]
	Auxiliary Task		[29]
	Network Architecture Optimization		[30] [31] [32]
	Octree-represented Voxel Grid		[33] [34] [35]
	Unsupervised Representation Learning		[36] [37] [38] [39]
	Non-convolutional Approaches		[40]
	Conversion from a Point Cloud		[25] [41] [42]
Multi-view (images)	Basic Architectures		[43] [44] [13]
	Multiple Modalities		[45] [46]
	Axis-aligned Views		[47] [48]
	Learned View Grouping		[49] [50]
	Unsupervised Viewpoints Assignment		[51] [52]
	Unsupervised Representation Learning		[53]
	Using Auxiliary Data		[54]
	Special Projections	Geometry Images	[55]
		Panorama	[56] [57] [58]
		Spherical	[59]
Point cloud	Symmetric Operation on Points		[60] [61]
	Hierarchical Feature Extraction		[62] [63] [64]
	Convolution on Neighborhood Graph		[65] [66]
	Convolution on Points	Grid Around the Query Point	[67] [68] [69]
		Continuous Convolution	[70] [71] [72] [73] [74] [75] [76]
	Sequential Processing	Using Attention Mechanism	[77]
		Encoding Locality into Order	[78]
	Unsupervised Learning of Shapes		[79] [80]
Surface shape	Manifold-based Convolution		[81] [82] [83]
	Graph-based Convolution		[84] [85] [86] [87]
	Native Mesh-based Approaches		[88] [89] [90] [91]
Hybrid	Ensembling		[92] [13] [20]
	Descriptor Merging		[93]

Figure 4-10. Taxonomy of The Approaches

The datasets used for the evaluation are: ModelNet40, aligned ModelNet40, and ShapeNetCore. The experiments are conducted on Linux machines with AMD RYZEN 1950X or two Intel Xeon E5-2680 v3 CPUs, 128 GB of RAM and NVIDIA GeForce GTX 1080 Ti GPUs with driver version 440.44.

As results, during the training, the reported accuracies were not reached. The main reason appears to be the usage of different data conversion methods. The researchers found that the data conversion method (e.g., point cloud sampling or image rendering method) can significantly impact the classification accuracy. They observe multi-view image-based representations yielding the best classification accuracy and rotational alignment being beneficial to mainly point-cloud-based networks. A larger dataset can also improve accuracies of most networks, especially on image-based networks.

4.5.2 Deep Learning for 3D Shape Classification from Multiple Depth Maps

This paper proposes a novel approach for the classification of 3D shapes exploiting a multi-branch Convolutional Neural Network (CNN). The algorithm starts by constructing a set of depth maps by rendering the input 3D shape from different viewpoints.

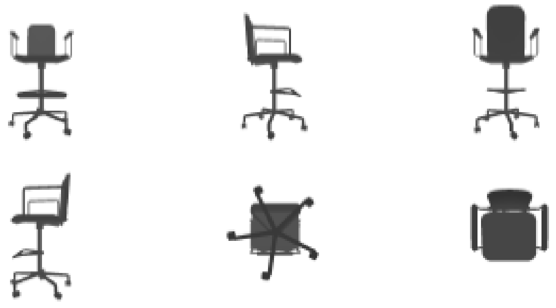


Figure 4-11. Example of 6 Depth Maps of A Chair

The depth maps are fed to a multi-branch Convolutional Neural Network. Each branch of the network takes in input one of the depth maps and produces a classification vector by using 5 convolutional layers of progressively reduced resolution.

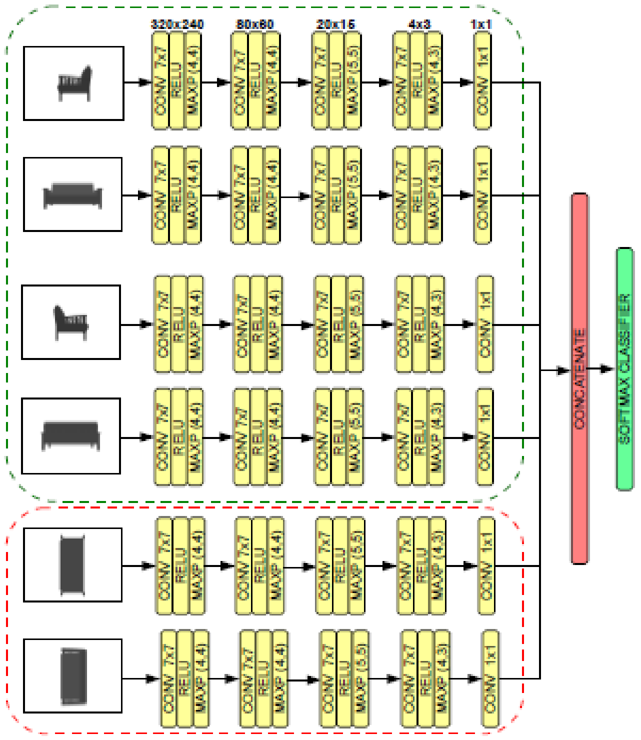
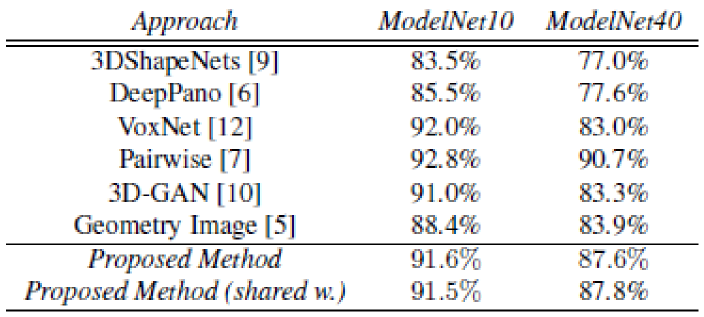


Figure 4-12. Architecture of The CNN

The various classification vectors are finally fed to a linear classifier that combines the outputs of the various branches and produces the final classification.

The dataset used in this research is the Princeton ModelNet. Two subsets are used, namely: the ModelNet10 subset which contains 4,899 3D models divided into 10 different categories and the ModelNet40 subset which contains 12,311 3D models divided into 40 different categories.



Approach	*ModelNet10*	*ModelNet40*
3DShapeNets [9]	83.5%	77.0%
DeepPano [6]	85.5%	77.6%
VoxNet [12]	92.0%	83.0%
Pairwise [7]	92.8%	90.7%
3D-GAN [10]	91.0%	83.3%
Geometry Image [5]	88.4%	83.9%
Proposed Method	91.6%	87.6%
Proposed Method (shared w.)	91.5%	87.8%

Figure 4-13. Average Accuracies

According to the researchers, the proposed approach is fast and requires a relatively small training effort, especially if the proposed weights sharing approach is applied, The performance of the proposed network outperforms several recent state-of-the-art approaches, as shown in Figure 4-13 above.

4.5.3 PointHop: An Explainable Machine Learning Method for Point Cloud Classification

This paper proposed a method called PointHop for point cloud classification in Machine Learning. Point cloud models are popular due to easy access and complete description in the 3D space. A point cloud is represented by a set of points in the 3D coordinates.

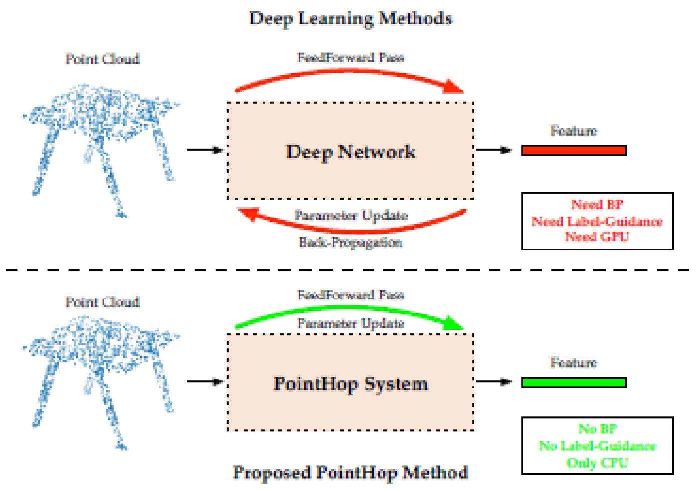


Figure 4-14. Comparison of Existing Methods and PointHop

The PointHop method consists of two stages:

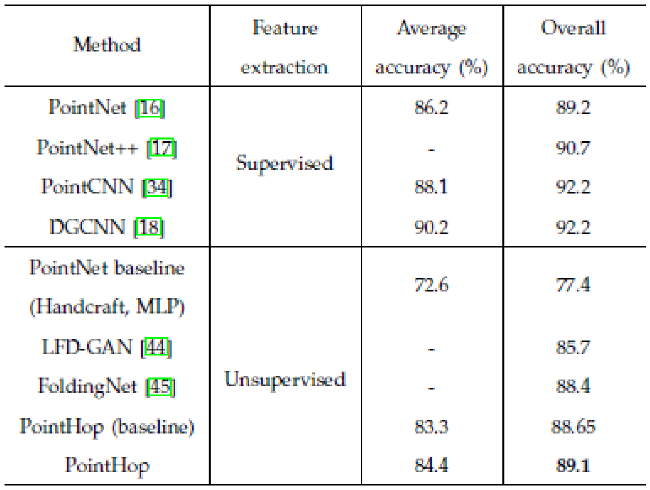
1. Local-to-global attribute building through iterative one-hop information exchange.

   In the attribute building stage, the problem of unordered point cloud data is addressed using a space partitioning procedure and by developing an effective and robust descriptor that characterizes the relationship between a point and its one-hop neighbor in a PointHop unit.

2. Classification and ensembles.

The feature vector obtained from multiple PointHop units is fed to a classifier, such as the support vector machine (SVM) classifier and the random forest (RF) classifier to get classification result.

The dataset used in this research is the ModelNet40. The dataset contains 40 categories of CAD models of objects such as airplanes, chairs, benches, cups, etc. Each initial point cloud has 2,048 points and each point has three Cartesian coordinates. There are 9,843 training samples and 2,468 testing samples.



Method	Feature extraction	Average accuracy (%)	Overall accuracy (%)
PointNet [16]	Supervised	86.2	89.2
PointNet++ [17]		-	90.7
PointCNN [34]		88.1	92.2
DGCNN [18]		90.2	92.2
PointNet baseline (Handcraft, MLP)	Unsupervised	72.6	77.4
LFD-GAN [44]		-	85.7
FoldingNet [45]		-	88.4
PointHop (baseline)		83.3	88.65
PointHop		84.4	**89.1**

Figure 4-15. Accuracies Comparison

During the experiment, the PointHop method significantly reduces the training process, while still maintaining the classification performance, in comparison with the state-of-the-art Deep Learning.

5 Methodology

5.1 Overview

This project on computer vision consists of two studies in the field of 3D Deep Learning. The first one is to examine the classification problem of 3D models and then the second one is to examine the generation or reconstruction problem of 3D models.

This project uses Python as the main programming language. In term of the Deep Learning library, both frameworks mentioned in the previous chapter, TensorFlow and PyTorch, are used interchangeably.

The machines used in this project are as follows:

- Windows machine:

• Processor:	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 2112 Mhz 4 Core(s), 8 Logical Processor(s)
• RAM:	32 GB
• GPU:	- Intel(R) UHD Graphics 620 (internal) - NVIDIA Quadro P500
• eGPU:	NVIDIA GeForce GTX 1080 Ti

- Mac machine:

• Processor:	Apple M1 SoC 8-Core CPU, 16-Core Neural Engine 4 cores high-performance, 4 cores energy-efficient
• RAM:	8 GB
• GPU:	7-Core (integrated)

Development Environment

Anaconda[5] is used as the platform to help the development of Machine Learning. It provides almost all the necessary packages for Machine Learning in Python. As for the editor, Jupyter Notebook and JupyterLab are used. They are simple and stable, especially for Machine Learning prototyping purposes.

Due to the limited capabilities of the machines, Google Colab[6] is also used in the experiment.

Microsoft Visual Studio Code is also used to program fast required Python components.

In Python development, besides the standard Machine Learning libraries, additional libraries are used. The important ones are Trimesh used for 3D object operations, Glob used for directory operations, and PPTK used for point processing.

[5] https://www.anaconda.com/

[6] https://colab.research.google.com/

5.2 Dataset

The main dataset used in this project is sourced from the Princeton ModelNet[7]. The Princeton ModelNet contains collection of 3D CAD models for objects. The 3D CAD models are categorized. There are two subsets of the Princeton ModelNet dataset, namely ModelNet10 and ModelNet40. The file format is Object File Format (OFF). This is a geometry definition file format and can store 2D or 3D objects.

Figure 5-1. Samples from ModelNet

ModelNet10

The ModelNet10 subset contains CAD models in 10 categories, namely:

- bathtub
- bed
- chair
- desk
- dresser
- monitor
- night_stand
- sofa
- table
- toilet

There are 4,899 models in total, they are split into 3,991 models for training set and 908 model for test set.

For the multi-view approach, rendered images from 12 perspectives are used.

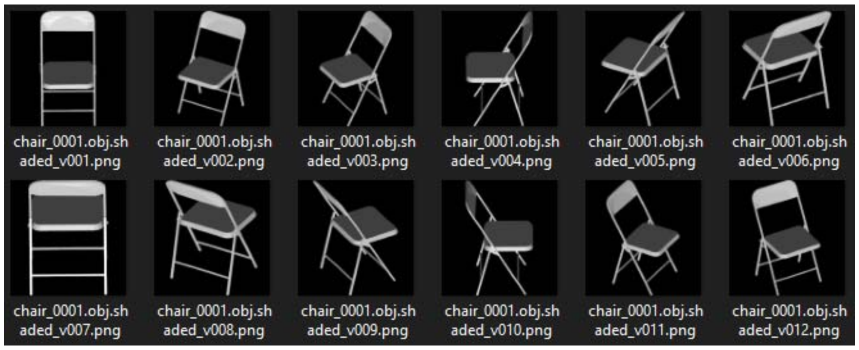

chair_0001.obj.shaded_v001.png
chair_0001.obj.shaded_v002.png
chair_0001.obj.shaded_v003.png
chair_0001.obj.shaded_v004.png
chair_0001.obj.shaded_v005.png
chair_0001.obj.shaded_v006.png
chair_0001.obj.shaded_v007.png
chair_0001.obj.shaded_v008.png
chair_0001.obj.shaded_v009.png
chair_0001.obj.shaded_v010.png
chair_0001.obj.shaded_v011.png
chair_0001.obj.shaded_v012.png

Figure 5-2. A Chair Rendered in 12 Perspectives

ModelNet40

The ModelNet40 subset contains CAD models in 40 categories, namely:

- airplane
- bathtub
- bed
- bench
- bookshelf
- bottle
- bowl
- chair
- cone
- cup
- curtain
- desk
- door
- dresser
- glass_box
- guitar
- keyboard
- lamp
- laptop
- mantel
- monitor
- person
- piano
- plant
- radio
- range_hood
- sink
- sofa
- stool
- table
- tent
- toilet
- tv_stand
- vase
- wardrobe

[7] https://modelnet.cs.princeton.edu/

• car	• flower_pot	• night_stand	• stairs	• xbox

There are 12,311 models in total, they are split into 9,843 models for training set and 2,468 model for test set.

Same as the ImageNet10, for the multi-view approach, rendered images from 12 perspectives are used.

Additionally, the CIFAR-10[8] dataset is also used to verify some basic functionalities in classification using 2D images. The CIFAR-10 dataset consists of 60,000 colour images with 32x32 size, categorized in 10 classes. Each class consists of 6,000 images. They are split into 50,000 training images and 10,000 test images.

5.3 Classification Problem

Classification problem becomes the "first-step" problem when it comes to Deep Learning, similar to the "Hello, World" when it comes to programming.

Two approaches are used for the classification in this project. The first approach is using multi-view images and the second one is using point-cloud generated on 3D objects.

5.3.1 Multi-view Images Method

Basically, classification 3D objects using multi-view images is quite similar to standard 2D images classification. The first experiment is 2D classification on CIFAR-10. The purpose of this first experiment is to make sure that the environment setup is adequate. The Deep Learning framework used in this experiment is TensorFlow.

TensorFlow must be available on the einvironment setup. Import TensorFlow and confirm the version:

```
import tensorflow as tf
tf_version = tf.__version__

print(tf_version)
```

5.3.1.1 Introduction on CIFAR-10

- Load and prepare the CIFAR-10 dataset

  Loading the CIFAR-10 dataset is quite straightforward because it is included in Keras.

```
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

- Inspect and verify the data

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
```

[8] https://www.cs.toronto.edu/~kriz/cifar.html

```
    plt.grid(False)
    plt.imshow(X_train[i])
    plt.xlabel(class_names[y_train[i][0]])
plt.show()
```

The images and the labels are displayed as follows:

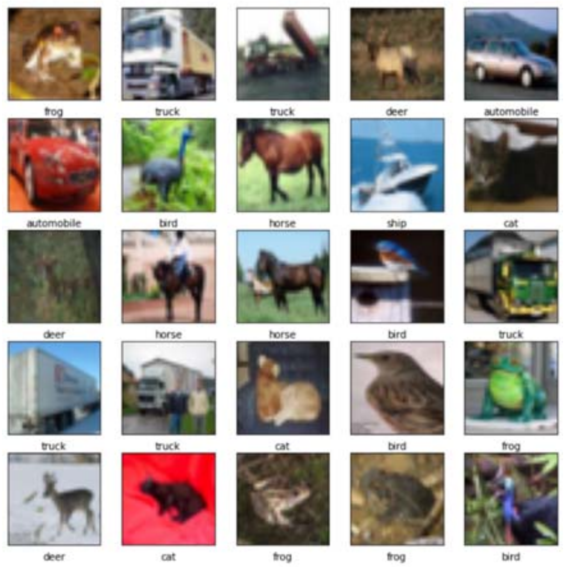


Figure 5-3. CIFAR-10 Images and Labels

- Preprocess the data

  The input layer will be flatten and the pixel values are normalize in the floating number of range of 0.0 to 1.0.

```
X_train =  X_train.reshape(50000, 32*32*3)  #change the shape towards (50000, 32*32*3)
X_test =  X_test.reshape(10000, 32*32*3)    #(10000, 32*32*3)
X_train =  X_train.astype('float32')        #change the type towards float32
X_test = X_test.astype('float32')           #change the type towards float32
X_train /= 255.0                            #normalize the range to be between 0.0 and 1.0
X_test /= 255.0                             #normalize the range to be between 0.0 and 1.0
```

- Create the model

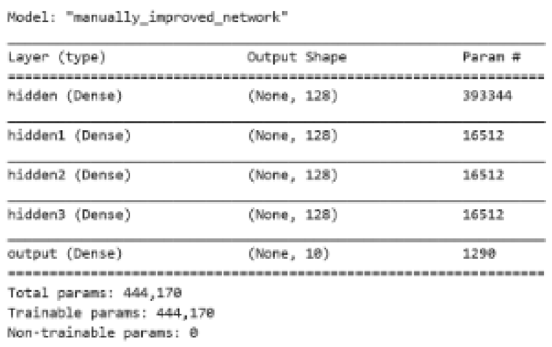
  Simple sequential model is created using 4 hidden dense layers, which mean all nodes between layers are connected. Each layer has 128 node. As for the activation, softmax is used.

```
epochs = 50
batches = 128
D = X_train.shape[1]
H = 128

model4 = Sequential(name='manually_improved_network')

model4.add(Dense(H, input_shape=(D,), activation='relu', name="hidden"))
model4.add(Dense(H, activation='relu', name="hidden1"))
model4.add(Dense(H, activation='relu', name="hidden2"))
model4.add(Dense(H, activation='relu', name="hidden3"))
model4.add(Dense(n_classes, input_shape=(D,), activation='softmax', name="output"))
```

The architecture of the model based on above implementation is summarized as follows:



```
Model: "manually_improved_network"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden (Dense)               (None, 128)               393344
_________________________________________________________________
hidden1 (Dense)              (None, 128)               16512
_________________________________________________________________
hidden2 (Dense)              (None, 128)               16512
_________________________________________________________________
hidden3 (Dense)              (None, 128)               16512
_________________________________________________________________
output (Dense)               (None, 10)                1290
=================================================================
Total params: 444,170
Trainable params: 444,170
Non-trainable params: 0
```

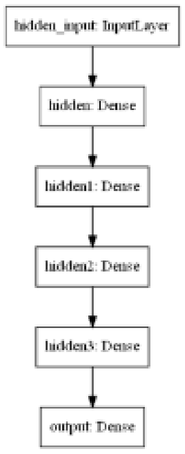


Figure 5-4. Model Architecture for CIFAR-10 Case

- Compile and train the model

```
model4.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
             optimizer='adam',
             metrics=['accuracy'])

log4 = model4.fit(X_train,
                 Y_train,
                 batch_size=batches,
                 epochs=epochs,
                 validation_data=(X_test, Y_test))
```

- Result and Evaluation

```
f = plt.figure(figsize=(12,4))
ax1 = f.add_subplot(121)
ax2 = f.add_subplot(122)
ax1.plot(log4.history['loss'], label='Training loss')
ax1.plot(log4.history['val_loss'], label='Testing loss')
ax1.legend()
ax1.grid()
ax2.plot(log4.history['accuracy'], label='Training acc')
ax2.plot(log4.history['val_accuracy'], label='Val acc')
ax2.legend()
ax2.grid()

loss_test, metric_test = model4.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', loss_test)
print('Test accuracy:', metric_test)
```

```
Test loss: 1.7683160305023193
Test accuracy: 0.4927999973297119
```

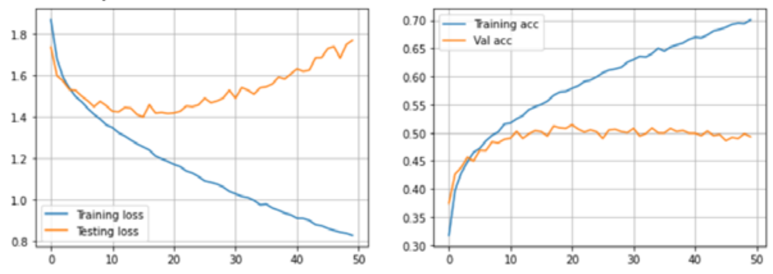


Figure 5-5. Evaluation on CIFAR-10 Case

The result is unsurprisingly bad, because the model is quite naively constructed using only dense layers. The test dataset is also used for both validation and evaluation, which is not the best practice. The test accuracy is only 49.28%. But it serves the purpose to provide short and quick confirmation of the environment setup.

5.3.1.2 Experiment on ModelNet10

5.3.1.2.1 First Experiment

- Load and prepare the CIFAR-10 dataset

  The ModelNet10 has two subset, training and test. For the validation during the training, 20% of the training set is used. The test set will only used for evaluation.

```
import os

IMG_HEIGHT = 224
IMG_WIDTH = 224
BATCH_SIZE = 16

# Load data from directory
data_dir = 'data/modelnet10_images'

train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(data_dir, "train"),
    validation_split = 0.2,
    subset = "training",
    seed = 123,
    image_size = (IMG_HEIGHT, IMG_WIDTH),
    batch_size = BATCH_SIZE)
val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(data_dir, "train"),
    validation_split = 0.2,
    subset = "validation",
    seed = 123,
    image_size = (IMG_HEIGHT, IMG_WIDTH),
    batch_size = BATCH_SIZE)
test_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(data_dir, "test"),
    seed = 123,
    image_size = (IMG_HEIGHT, IMG_WIDTH),
    batch_size = BATCH_SIZE)
```

- Inspect and verify the data

```
import matplotlib.pyplot as plt

def show_imgs(dataset, nr):
    plt.figure(figsize = (10, 10))
    for images, labels in dataset.take(1):
        for i in range(nr):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(class_names[labels[i]])
            plt.axis("off")

print("Train dataset:")
show_imgs(train_dataset, 9)
```

The images and the labels are displayed as follows:

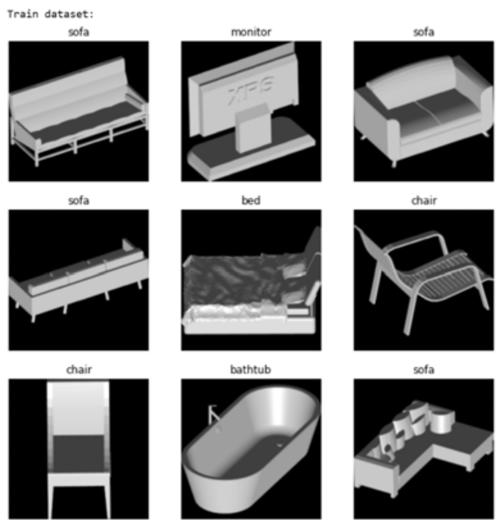


Figure 5-6. ImageNet10 Images and Labels

- Preprocess the data

  For the normalization, rescaling layer from Keras is used.

```
from tensorflow.keras.layers.experimental.preprocessing import Rescaling

normalization_layer = Rescaling(1./255)
```

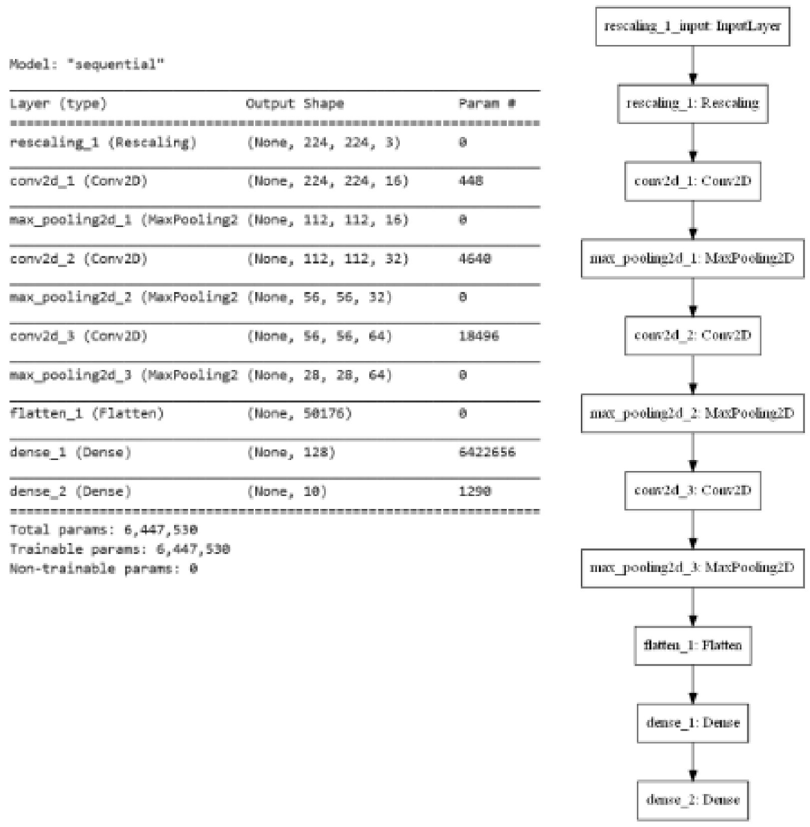
  This normalization layer can be added directly during the creation of the model.

- Create the model

  The normalization of the images is added directly after the input. Relu is used as the activation function.

```
model0 = Sequential([
  Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(n_classes)
])
```

  The architecture of the model based on above implementation is summarized as follows:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 224, 224, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               6422656
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 6,447,530
Trainable params: 6,447,530
Non-trainable params: 0
```

rescaling_1_input: InputLayer
rescaling_1: Rescaling
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_2: Conv2D
max_pooling2d_2: MaxPooling2D
conv2d_3: Conv2D
max_pooling2d_3: MaxPooling2D
flatten_1: Flatten
dense_1: Dense
dense_2: Dense

Figure 5-7. First Model Architecture for ModelNet10

- Compile and train the model

  The number of epochs is set to 5 and the optimizer is adam.

```
model0.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

log = model0.fit(train_dataset,
                epochs=5,
                validation_data=val_dataset)
```

- Result and Evaluation

  In the training process, the training accuracy reached 98.84% and the validation accuracy stayed in the range of 90-95%.

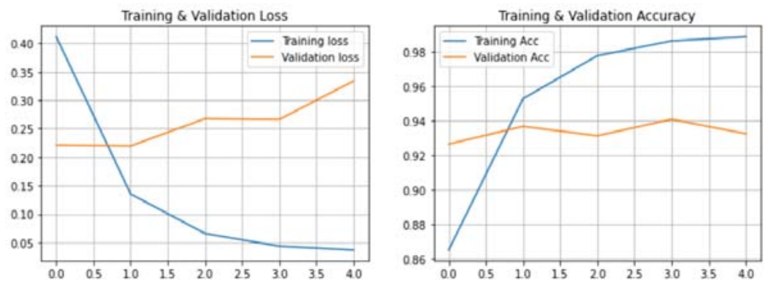
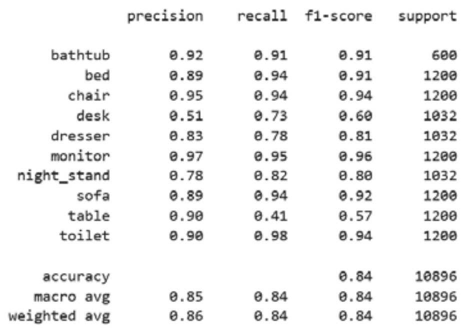


Figure 5-8. First Model Training & Validation on ModelNet10

Evaluation of the trained model on the test dataset gives a very good result. Test accuracy is 83.95%. Following is the classification report:



```
              precision    recall  f1-score   support

     bathtub       0.92      0.91      0.91       600
         bed       0.89      0.94      0.91      1200
       chair       0.95      0.94      0.94      1200
        desk       0.51      0.73      0.60      1032
     dresser       0.83      0.78      0.81      1032
     monitor       0.97      0.95      0.96      1200
 night_stand       0.78      0.82      0.80      1032
        sofa       0.89      0.94      0.92      1200
       table       0.90      0.41      0.57      1200
      toilet       0.90      0.98      0.94      1200

    accuracy                           0.84     10896
   macro avg       0.85      0.84      0.84     10896
weighted avg       0.86      0.84      0.84     10896
```

Figure 5-9. First Model Classification Report for ModelNet10

5.3.1.2.2 Second Experiment

The second experiment use more complex model. Transfer learning using MobileNetV2 is applied. MobileNet is Convolutional Neural Networks (CNN) which is basically developed for mobile vision application. It has the classification models and all the pre-trained weights.

- Create the model

  The implementation into the experiment is as follows:

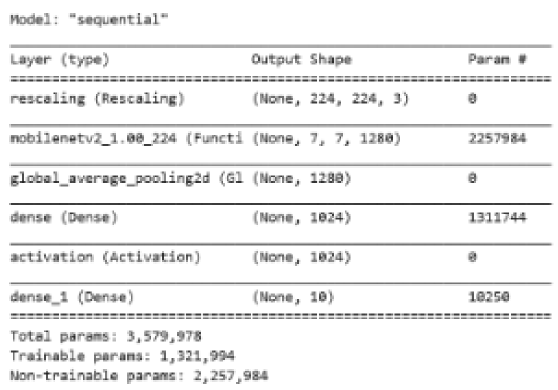
```
from tensorflow.keras.layers.experimental.preprocessing import Rescaling

base_model = keras.applications.MobileNetV2(
    weights='imagenet',
    include_top=False,
    input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

# Freeze the base_model
base_model.trainable = False

# Create the model structure
model = tf.keras.Sequential([
    keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, 3)),
    Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    base_model,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(1024),
    keras.layers.Activation('relu'),
    keras.layers.Dense(n_classes)
])
```

The pre-trained weights are based on ImageNet. In this experiment the imported model is freezed, this means that the existing internal structure will not be changed in the training.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 224, 224, 3)       0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1311744
_________________________________________________________________
activation (Activation)      (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 3,579,978
Trainable params: 1,321,994
Non-trainable params: 2,257,984
```

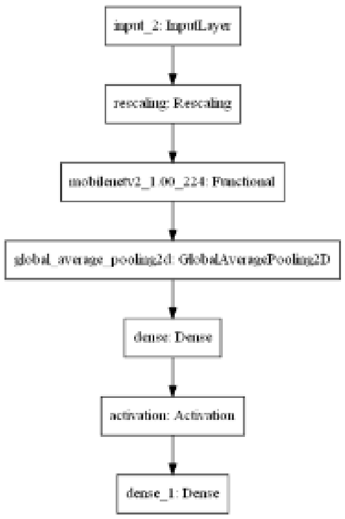


Figure 5-10. Second Model Architecture for ModelNet10

- Result and Evaluation

  In the training process, the training accuracy reached 97.01% and the validation accuracy reached 95.12%.

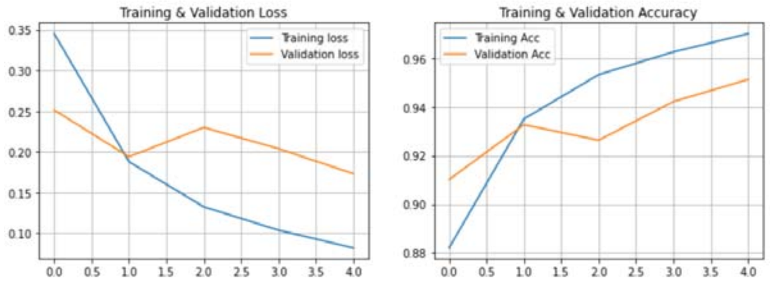
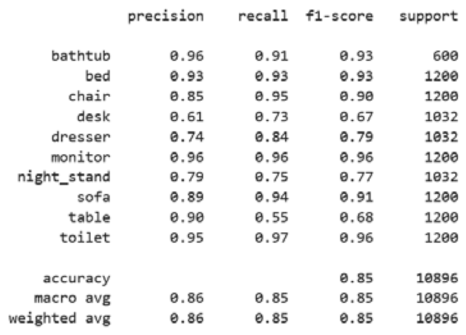


Figure 5-11. Second Model Training & Validation on ModelNet10

Evaluation of the second model on the test dataset gives a better result than the first one. Test accuracy is 85.34%. Following is the classification report:



```
              precision    recall  f1-score   support

     bathtub       0.96      0.91      0.93       600
         bed       0.93      0.93      0.93      1200
       chair       0.85      0.95      0.90      1200
        desk       0.61      0.73      0.67      1032
     dresser       0.74      0.84      0.79      1032
     monitor       0.96      0.96      0.96      1200
 night_stand       0.79      0.75      0.77      1032
        sofa       0.89      0.94      0.91      1200
       table       0.90      0.55      0.68      1200
      toilet       0.95      0.97      0.96      1200

    accuracy                           0.85     10896
   macro avg       0.86      0.85      0.85     10896
weighted avg       0.86      0.85      0.85     10896
```

Figure 5-12. Second Model Classification Report for ModelNet10

5.3.1.3 Experiment on ModelNet40

5.3.1.3.1 First Experiment

Using the same approaches and model from the first experiment on ModelNet40 gives lower results.

In the training process, the training accuracy reached 97.79% and the validation accuracy reached 88.15%.

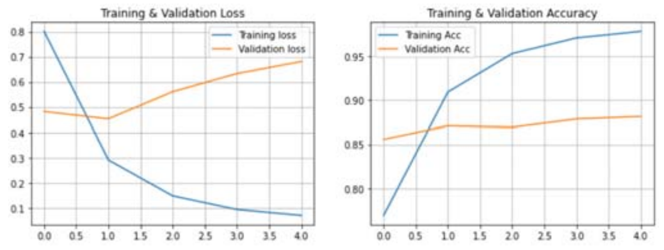
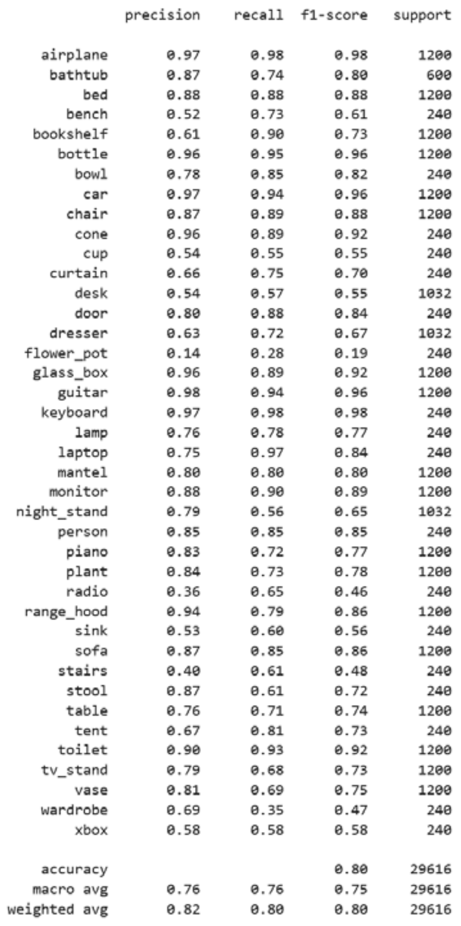


Figure 5-13. First Model Training & Validation on ModelNet40

Evaluation of the first model on the test dataset gives also a lower result compared to ModelNet10. Test accuracy is 79.70%. Following is the classification report:



```
              precision    recall  f1-score   support

    airplane       0.97      0.98      0.98      1200
     bathtub       0.87      0.74      0.80       600
         bed       0.88      0.88      0.88      1200
       bench       0.52      0.73      0.61       240
   bookshelf       0.61      0.90      0.73      1200
      bottle       0.96      0.95      0.96      1200
        bowl       0.78      0.85      0.82       240
         car       0.97      0.94      0.96      1200
       chair       0.87      0.89      0.88      1200
        cone       0.96      0.89      0.92       240
         cup       0.54      0.55      0.55       240
     curtain       0.66      0.75      0.70       240
        desk       0.54      0.57      0.55      1032
        door       0.80      0.88      0.84       240
     dresser       0.63      0.72      0.67      1032
  flower_pot       0.14      0.28      0.19       240
   glass_box       0.96      0.89      0.92      1200
      guitar       0.98      0.94      0.96      1200
    keyboard       0.97      0.98      0.98       240
        lamp       0.76      0.78      0.77       240
      laptop       0.75      0.97      0.84       240
      mantel       0.80      0.80      0.80      1200
     monitor       0.88      0.90      0.89      1200
 night_stand       0.79      0.56      0.65      1032
      person       0.85      0.85      0.85       240
       piano       0.83      0.72      0.77      1200
       plant       0.84      0.73      0.78      1200
       radio       0.36      0.65      0.46       240
  range_hood       0.94      0.79      0.86      1200
        sink       0.53      0.60      0.56       240
        sofa       0.87      0.85      0.86      1200
      stairs       0.40      0.61      0.48       240
       stool       0.87      0.61      0.72       240
       table       0.76      0.71      0.74      1200
        tent       0.67      0.81      0.73       240
      toilet       0.90      0.93      0.92      1200
    tv_stand       0.79      0.68      0.73      1200
        vase       0.81      0.69      0.75      1200
    wardrobe       0.69      0.35      0.47       240
        xbox       0.58      0.58      0.58       240

    accuracy                           0.80     29616
   macro avg       0.76      0.76      0.75     29616
weighted avg       0.82      0.80      0.80     29616
```

Figure 5-14. First Model Classification Report for ModelNet40

5.3.1.3.2 Second Experiment

Using the same transfer learning approaches on MobileNetV2 from the second experiment on ModelNet40 gives also lower results. But it is similarly better than the application of the first experiment.

In the training process, the training accuracy reached 94.34% and the validation accuracy reached 90.92%.

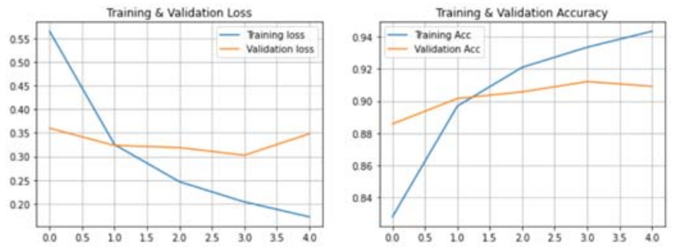
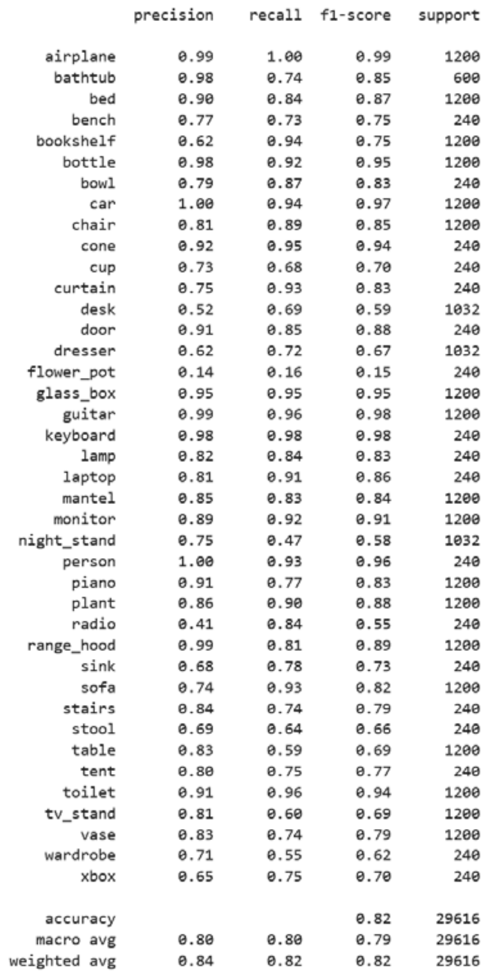


Figure 5-15. Second Model Training & Validation on ModelNet40

Evaluation of the second model on the test dataset gives also a lower result compared to ModelNet10 but better than the application of first model on ModelNet40. Test accuracy is 82.04%. Following is the classification report:



```
              precision    recall  f1-score   support

    airplane       0.99      1.00      0.99      1200
     bathtub       0.98      0.74      0.85       600
         bed       0.90      0.84      0.87      1200
       bench       0.77      0.73      0.75       240
   bookshelf       0.62      0.94      0.75      1200
      bottle       0.98      0.92      0.95      1200
        bowl       0.79      0.87      0.83       240
         car       1.00      0.94      0.97      1200
       chair       0.81      0.89      0.85      1200
        cone       0.92      0.95      0.94       240
         cup       0.73      0.68      0.70       240
     curtain       0.75      0.93      0.83       240
        desk       0.52      0.69      0.59      1032
        door       0.91      0.85      0.88       240
     dresser       0.62      0.72      0.67      1032
  flower_pot       0.14      0.16      0.15       240
   glass_box       0.95      0.95      0.95      1200
      guitar       0.99      0.96      0.98      1200
    keyboard       0.98      0.98      0.98       240
        lamp       0.82      0.84      0.83       240
      laptop       0.81      0.91      0.86       240
      mantel       0.85      0.83      0.84      1200
     monitor       0.89      0.92      0.91      1200
 night_stand       0.75      0.47      0.58      1032
      person       1.00      0.93      0.96       240
       piano       0.91      0.77      0.83      1200
       plant       0.86      0.90      0.88      1200
       radio       0.41      0.84      0.55       240
  range_hood       0.99      0.81      0.89      1200
        sink       0.68      0.78      0.73       240
        sofa       0.74      0.93      0.82      1200
      stairs       0.84      0.74      0.79       240
       stool       0.69      0.64      0.66       240
       table       0.83      0.59      0.69      1200
        tent       0.80      0.75      0.77       240
      toilet       0.91      0.96      0.94      1200
    tv_stand       0.81      0.60      0.69      1200
        vase       0.83      0.74      0.79      1200
    wardrobe       0.71      0.55      0.62       240
        xbox       0.65      0.75      0.70       240

    accuracy                           0.82     29616
   macro avg       0.80      0.80      0.79     29616
weighted avg       0.84      0.82      0.82     29616
```

Figure 5-16. Second Model Classification Report for ModelNet40

5.3.2 Point-cloud Method

With the increase of the usage on sensors in 3D space, especially Lidar (Light Detection and Ranging), 3D objects classification using point-cloud becomes more important.

In the following experiment, classifications using point-cloud are performed. The first experiment uses Kaolin and PyTorch. The second one uses Keras and TensorFlow. Because of the limitation on computation capacity, both experiments are performed only on ModelNet10.

5.3.2.1 Using Kaolin and PyTorch

Kaolin provides operations for 3D objects and Deep Learning models. This experiment uses Kaolin for the generation of point-cloud and uses its PointNetClassifier for the classification problem. This experiment is performed in Google Colab.

- Load and preprocess the data

  The data is loaded using Kaolin's ModelNet. Therefore, the transformation from meshes into point-clouds can be integrated in one pass. For every meshes 1000 points will be generated.

```
datapath = DATA_DIR10
device='cuda'
num_of_points = 1000

def to_device(inp):
    inp.to(device)
    return inp

transform = tfs.Compose([
    to_device,
    tfs.TriangleMeshToPointCloud(num_samples=num_of_points),
    tfs.NormalizePointCloud()
])

num_of_workers = 0 if device == 'cuda' else 10
memory = device != 'cuda'
batch_size = 16

train_loader = DataLoader(ModelNet(datapath,
                                   categories=classes,
                                   split='train',
                                   transform=transform),
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=num_of_workers,
                          pin_memory=memory)

val_loader = DataLoader(ModelNet(datapath,
                                 categories=classes,
                                 split='test',
                                 transform=transform),
                        batch_size=batch_size,
                        num_workers=num_of_workers,
                        pin_memory=memory)
```

- Create the model

```
lr = 0.01

model = PointNetClassifier(num_classes=len(classes)).to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=lr)
criterion = torch.nn.CrossEntropyLoss()
```

- Train the model

  The training process is done in 5 ephocs and adam is used as the optimizer.

```
epochs = 5

for e in range(epochs):
    print(f'{"":=<25}\nEpoch: {e+1}\n')

    train_loss = 0.0
    train_accuracy = 0.0

    model.train()
    for batch_idx, (data, attributes) in enumerate(tqdm(train_loader)):
        category = attributes['category'].to(device)
        pred = model(data)
        loss = criterion(pred, category.view(-1))
        train_loss += loss.item()
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        # Accuracy
        pred_label = torch.argmax(pred, dim=1)
        train_accuracy += torch.mean((pred_label == category.view(-1)).float()).item()

    print('Train loss:', train_loss / len(train_loader))
    print('Train accuracy:', train_accuracy / len(train_loader))

    val_loss = 0.
    val_accuracy = 0.

    model.eval()
    with torch.no_grad():
        for batch_idx, (data, attributes) in enumerate(tqdm(val_loader)):
            category = attributes['category'].to(device)
            pred = model(data)
            loss = criterion(pred, category.view(-1))
            val_loss += loss.item()

            # Accuracy
            pred_label = torch.argmax(pred, dim=1)
            val_accuracy += torch.mean((pred_label == category.view(-1)).float()).item()

    print('Val loss:', val_loss / len(val_loader))
    print('Val accuracy:', val_accuracy / len(val_loader))
```

- Result and Evaluation

  In the training process, the training accuracy reached 88.09% and the validation accuracy reached 77.52%. The validation uses the test dataset.

  As for the evaluation of the trained model, the test dataset is used again, but it is shuffled. Figure 5-17 is the visualization of some evaluated point-clouds. The correct identified or classified objects (point-clouds) are shown in green and the wrong ones are in red.

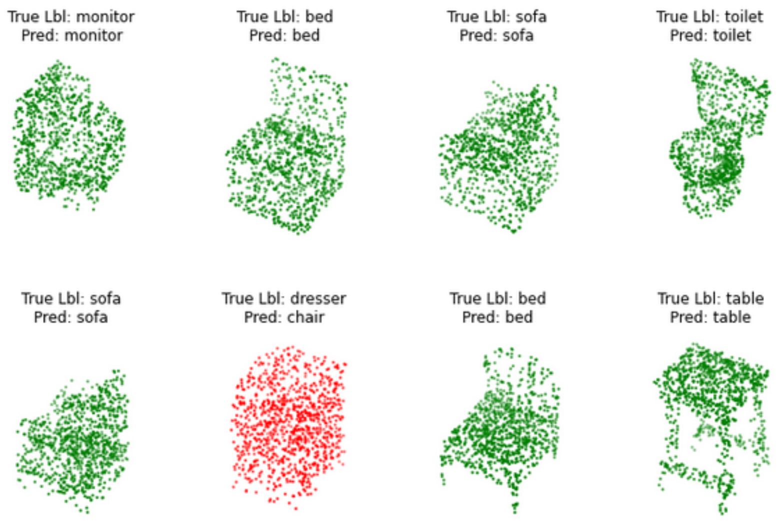


Figure 5-17. Point-cloud Classification on Kaolin & PyTorch

5.3.2.2 Using Keras and TensorFlow

- Load and preprocess the data

  For the transformation from meshes to points, Trimesh function sample() is used. The number of sampled points is 1000.

```
def transform_dataset(data_dir = "", num_points = 1000):
    train_points = []
    train_labels = []
    test_points = []
    test_labels = []
    class_map = {}
    folders = glob.glob(os.path.join(data_dir, "*"))

    for i, folder in enumerate(folders):
        print("Processing class: {}".format(os.path.basename(folder)))
        # store folder name with ID so we can retrieve later
        class_map[i] = folder.split("/")[-1]
        # gather all files
        train_files = glob.glob(os.path.join(folder, "train/*"))
        test_files = glob.glob(os.path.join(folder, "test/*"))

        for f in train_files:
            train_points.append(trimesh.load(f).sample(num_points))
            train_labels.append(i)

        for f in test_files:
            test_points.append(trimesh.load(f).sample(num_points))
            test_labels.append(i)

    return (
        np.array(train_points),
        np.array(test_points),
        np.array(train_labels),
        np.array(test_labels),
        class_map,
    )

NUM_POINTS = 1000
DATA_DIR = 'data/ModelNet10'
train_points, test_points, train_labels, test_labels, class_map = transform_dataset(DATA_DIR,
NUM_POINTS)
```

- Augmenting and Packing into Datasets

```
BATCH_SIZE = 16

train_dataset = tf.data.Dataset.from_tensor_slices((train_points, train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices((test_points, test_labels))

train_dataset = train_dataset.shuffle(len(train_points)).map(augment).batch(BATCH_SIZE)
test_dataset = test_dataset.shuffle(len(test_points)).batch(BATCH_SIZE)
```

- Create the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras import regularizers

NUM_CLASSES = 10

# Create the model structure
model = Sequential([
    layers.Input(shape=(NUM_POINTS, 3)),
    layers.Conv1D(32, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Conv1D(64, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Conv1D(512, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.GlobalMaxPooling1D(),
    layers.Dense(256),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Dense(128),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Dense(
        3 * 3,
        kernel_initializer="zeros",
        bias_initializer=keras.initializers.Constant(np.eye(3).flatten()),
        activity_regularizer=regularizers.L2(0.01)
    ),
    layers.Reshape((3, 3)),
    layers.Conv1D(32, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Conv1D(64, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Conv1D(512, kernel_size=1, padding="valid"),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.GlobalMaxPooling1D(),
    layers.Dense(256),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Dropout(0.2),
    layers.Dense(128),
    layers.BatchNormalization(momentum=0.0),
    layers.Activation("relu"),
    layers.Dropout(0.2),
    layers.Dense(NUM_CLASSES, activation="softmax")
])

model.summary()
```

- Compile and train the model

  The number of epochs is set to 20 and the optimizer is adam.

```
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["sparse_categorical_accuracy"],
)
epochs = 20

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

- Result and Evaluation

  The training process has the training accuracy of 84.31% and the validation accuracy of only 63%.

  The evaluation of the trained model is again performed to the test. Figure 5-18 is the visualization of some evaluated point-clouds. The correct identified or classified objects (point-clouds) are shown in green and the wrong ones are in red.

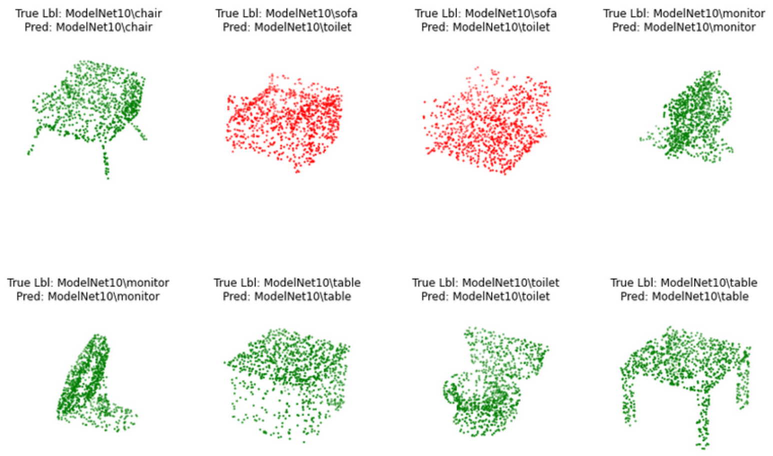

True Lbl: ModelNet10\chair
Pred: ModelNet10\chair
True Lbl: ModelNet10\sofa
Pred: ModelNet10\toilet
True Lbl: ModelNet10\sofa
Pred: ModelNet10\toilet
True Lbl: ModelNet10\monitor
Pred: ModelNet10\monitor
True Lbl: ModelNet10\monitor
Pred: ModelNet10\monitor
True Lbl: ModelNet10\table
Pred: ModelNet10\table
True Lbl: ModelNet10\toilet
Pred: ModelNet10\toilet
True Lbl: ModelNet10\table
Pred: ModelNet10\table

Figure 5-18. Point-cloud Classification on Keras & TensorFlow

5.4 Generation or Reconstruction Problem

There are many types and approaches for generation or reconstruction problem in 3D computer vision. This experiment examine the approach highlighted in PyTorch3D, namely the generation of 3D object using multi-view images. Because all models in ModelNet dataset are monotonous, this experiment uses the rendered silhouette images instead of the multi-view images.

The experiment on construction of a3D mesh using multi-view silhouette images contains following steps:

- Importing libraries and requirements

  Many PyTorch3D renderer functionalities are required for the experiment.

```
import os
import torch
import matplotlib.pyplot as plt

from pytorch3d.utils import ico_sphere
import numpy as np
import tqdm
import trimesh

# Util function for loading meshes
from pytorch3d.io import load_objs_as_meshes, save_obj, load_obj

from pytorch3d.loss import (
    chamfer_distance,
    mesh_edge_loss,
    mesh_laplacian_smoothing,
    mesh_normal_consistency,
)

# Data structures and functions for rendering
from pytorch3d.structures import Meshes
from pytorch3d.renderer import (
    look_at_view_transform,
    OpenGLPerspectiveCameras,
    PointLights,
    DirectionalLights,
    Materials,
    RasterizationSettings,
    MeshRenderer,
    MeshRasterizer,
    SoftPhongShader,
    SoftSilhouetteShader,
    SoftPhongShader,
    TexturesVertex
)

import sys

sys.path.append(os.path.abspath(''))
```

- Load and examine the object

  This experiment uses object from the ModelNet dataset. This dataset uses OFF format.

```
DATA_DIR = 'data/ModelNet40'

#obj_filename = os.path.join(DATA_DIR, "airplane/train/airplane_0001.off")
#obj_filename = os.path.join(DATA_DIR, "bathtub/train/bathtub_0005.off")
#obj_filename = os.path.join(DATA_DIR, "bottle/train/bottle_0003.off")
obj_filename = os.path.join(DATA_DIR, "chair/train/chair_0001.off")
#obj_filename = os.path.join(DATA_DIR, "laptop/train/laptop_0001.off")

# Use trimesh to visualize the data
target_mesh = trimesh.load_mesh(obj_filename)

target_mesh.show()
```

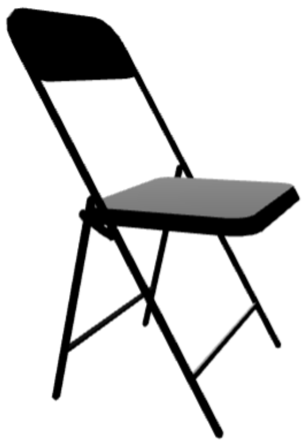


Figure 5-19. The Object chair_0001.off

- Preprocess the target object

  We will extract the mesh information from the object and create the PyTorch3D mesh from that. The created mesh needs to be normalized. The normalization includes resizing to fit for sphere with radius 1 and centering.

```
verts = torch.tensor(target_mesh.vertices)
verts = verts.type(torch.FloatTensor)
faces = torch.tensor(target_mesh.faces)

# Initialize each vertex to be black in color.
#verts_rgb = torch.zeros_like(verts)[None]  # (1, V, 3)
verts_rgb = torch.ones_like(verts)[None]  # (1, V, 3)
textures = TexturesVertex(verts_features=verts_rgb.to(device))

mesh = Meshes(
    verts=[verts.to(device)],
    faces=[faces.to(device)],
    textures=textures
)

verts = mesh.verts_packed()
N = verts.shape[0]
center = verts.mean(0)
scale = max((verts - center).abs().max(0)[0])
mesh.offset_verts_(-center)
mesh.scale_verts_((1.0 / float(scale)));
```

- Generate the multi-view silhouette images

  In this experiment, 40 images will be rendered from multiple viewpoints.

```
# Rasterization settings for silhouette rendering
sigma = 1e-4
raster_settings_silhouette = RasterizationSettings(
    image_size=128,
    blur_radius=np.log(1. / 1e-4 - 1.)*sigma,
    faces_per_pixel=50,
)

# Silhouette renderer
renderer_silhouette = MeshRenderer(
    rasterizer=MeshRasterizer(
        cameras=camera,
        raster_settings=raster_settings_silhouette
    ),
    shader=SoftSilhouetteShader()
)

# Render silhouette images.  The 3rd channel of the rendering output is
# the alpha/silhouette channel
silhouette_images = renderer_silhouette(meshes, cameras=cameras, lights=lights)
target_silhouette = [silhouette_images[i, ..., 3] for i in range(num_views)]

# Visualize silhouette images
image_grid(silhouette_images.cpu().numpy(), rows=8, cols=5, rgb=False)
plt.show()
```

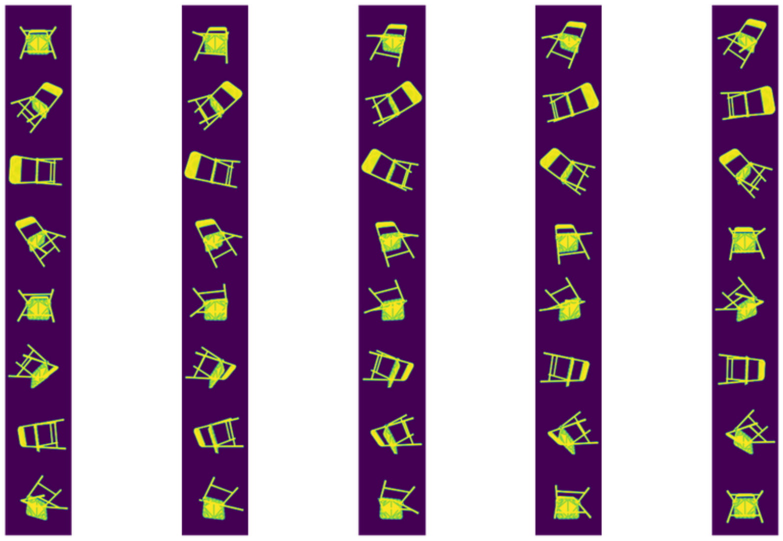


Figure 5-20. Rendered Silhouette Images

- Initialization for the mesh fitting

  The silhouette of the fitted mesh will be used to calculate loss. Therefore, new silhouette renderer is needed. As for the base mesh, a sphere of radius 1 is used. There are 2000 optimization steps used during the process.

```
# The source shape -> a sphere of radius 1.
src_mesh = ico_sphere(4, device)

# Renderer for Image-based 3D Reasoning', ICCV 2019
sigma = 1e-4
raster_settings_soft = RasterizationSettings(
    image_size=128,
    blur_radius=np.log(1. / 1e-4 - 1.)*sigma,
    faces_per_pixel=50,
)

# Silhouette renderer
renderer_silhouette = MeshRenderer(
    rasterizer=MeshRasterizer(
        cameras=camera,
        raster_settings=raster_settings_soft
    ),
    shader=SoftSilhouetteShader()
)
# Number of views to optimize over in each SGD iteration
num_views_per_iteration = 2
# Number of optimization steps
Niter = 2000
# Plot period for the losses
plot_period = 250

%matplotlib inline

losses = {"silhouette": {"weight": 1.0, "values": []},
          "edge": {"weight": 1.0, "values": []},
          "normal": {"weight": 0.01, "values": []},
          "laplacian": {"weight": 1.0, "values": []},
         }

# Losses to smooth / regularize the mesh shape
def update_mesh_shape_prior_losses(mesh, loss):
    # and (b) the edge length of the predicted mesh
    loss["edge"] = mesh_edge_loss(mesh)

    # mesh normal consistency
    loss["normal"] = mesh_normal_consistency(mesh)
    # mesh laplacian smoothing
    loss["laplacian"] = mesh_laplacian_smoothing(mesh, method="uniform")

verts_shape = src_mesh.verts_packed().shape
deform_verts = torch.full(verts_shape, 0.0, device=device, requires_grad=True)

# The optimizer
optimizer = torch.optim.SGD([deform_verts], lr=1.0, momentum=0.9)
```

- The mesh fitting

```
loop = tqdm(range(Niter))

for i in loop:
    # Initialize optimizer
    optimizer.zero_grad()

    # Deform the mesh
    new_src_mesh = src_mesh.offset_verts(deform_verts)

    # Losses to smooth /regularize the mesh shape
    loss = {k: torch.tensor(0.0, device=device) for k in losses}
    update_mesh_shape_prior_losses(new_src_mesh, loss)

    for j in np.random.permutation(num_views).tolist()[:num_views_per_iteration]:
        images_predicted = renderer_silhouette(new_src_mesh, cameras=target_cameras[j],
lights=lights)
        predicted_silhouette = images_predicted[..., 3]
        loss_silhouette = ((predicted_silhouette - target_silhouette[j]) ** 2).mean()
        loss["silhouette"] += loss_silhouette / num_views_per_iteration

    # Weighted sum of the losses
    sum_loss = torch.tensor(0.0, device=device)
    for k, l in loss.items():
        sum_loss += l * losses[k]["weight"]
        losses[k]["values"].append(float(l.detach().cpu()))

    # Print the losses
    loop.set_description("total_loss = %.6f" % sum_loss)

    # Plot mesh
    if i % plot_period == 0:
        visualize_prediction(new_src_mesh, title="iter: %d" % i, silhouette=True,
                             target_image=target_silhouette[1])
    sum_loss.backward()
    optimizer.step()
```

- Evaluation and Result

  The result is quite good. The problem occurs when the object has concave surfaces, because the silhouette has no information on that kind of surfaces.

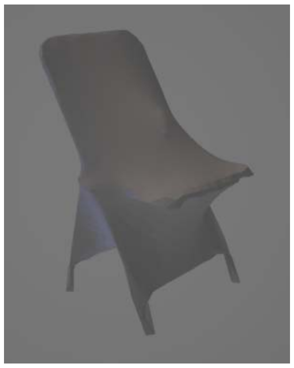


Figure 5-21. The Sculpted Mesh for chair_0001.off

6 Conclusions

Our world is in 3 dimensions (including time being 4 dimensions). Almost all objects are 3D. Projections of objects around us into our eyes are in 2D, but we have the ability to recognize the 3D shapes of these objects. This is currently what computer vision is trying to achieve with the help of machine learning and deep learning. The presence of additional sensors, such as the presence of Lidar (Light Detection and Ranging), reflects the use of our additional senses when we want to recognize 3D objects or scenes further.

This project studies how AI can recognize and reconstruct 3D objects. The current possibility, with the help of existing Deep Learning frameworks, such as TensorFlow and PyTorch, gives so much promise for further development and implementation. Using a relatively simple Deep Learning model with only a few learning, most of the neural networks models in this project are able to yield an average accuracy rate of over 60%. However, performance improvements can still be made by optimizing and fine-tuning the hyper parameters.

Another short-term strategy to gain performance is by applying the existing state-of-the-art Deep Learning model. In this project the state-of-the-art deep learning model has not been replicated, so the window for performance improvements is still very much open.

6.1 Lessons Learned

3D Deep Learning is still relatively new, but there are already many tools, libraries, frameworks, and models that can be used. We always have the option of building everything from scratch or developing on top of existing technology. But it's good in this case not to reinvent the wheel.

Even though machine learning looks like a black box, it is necessary to understand and have an intuition about how the box might operate. This is similar to what happens in our brains. That way, we can be better at preparing the data used for the learning process and can get an understanding of the level of accuracy produced.

Working with multiple applications, libraries, and devices requires careful attention to detail. Logs or records are needed, especially regarding version changes or configuration changes. It will be very time consuming if a problem arises that cannot be immediately identified whether it is related to the environment configuration or implementation.

Careful and meticulous planning is necessary, both on the project execution and on the documentation. Stay focused on the research, don’t get distracted, remember what the main goal is.

6.2 Encountered Problems

The main issue during the development of Machine Learning models is the computing resources. The machines used in this project have barely enough power for rough, short, and simple models on relatively small dataset, in this case the ModelNet10. As soon as the ModelNet40 is used, the training processes on the same models is taking significantly longer time to process.

Using Google Colab is actually very helpful, especially with the availability of GPU runtime. However, the default timeout from Google Colab is quite annoying, especially for training processes that take a long time. Sometimes the training process is stopped by the timeout and must be repeated from the first epoch. The usage of GPU power on the free version is also limited.

Another issue which happens quite often is the compatibility between libraries and APIs, especially when working on two different platforms. Although it is usually stated that the composition of several libraries is appropriate, sometimes the versions used are not suitable.

6.3 Future Work

Although Artificial Intelligence has been around for years, it can be said that it is still in its early stages of growth. In general, if AI is really an attempt to imitate the functions of the brain, then the gap in that direction is still very large, there are still many aspects that can be studied, applied, and utilized.

In the field of computer vision, both 2D and 3D, the use of AI, in this case Machine Learning and Deep Learning, has just started. From the internal point of view, the capability of existing AI models can still be improved. From the external point of view, there are still so many areas that have not taken advantage of AI yet.

This project is a starting point for more in-depth research on AI for 2D-3D computer vision. Further research on its actual use is still needed. The closest research that can be conducted is implementation and deployment of some of the Machine Learning models into real world applications. Further research on generative 3D design to construct editable 3D representation is also a possibility.

In the field of ClassCAD programming and the manufacturing industry in general, the exploration of the use of AI is still wide open. What is quite important is the collection of data needed for these studies. The AI-driven product development could also be an interesting topic in the next iteration.

7 List of Figures

8 References

Babu, A. D. (2019, November 5). *Artificial Intelligence vs Machine Learning vs Deep Learning (AI vs ML vs DL)*. Retrieved from Medium.com: https://medium.com/@alanb_73111/artificial-intelligence-vs-machine-learning-vs-deep-learning-ai-vs-ml-vs-dl-e6afb7177436

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., . . . Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012*.

Dubovikov, K. (2017, June 20). *PyTorch vs TensorFlow - Spotting The Difference*. Retrieved from Towards Data Science: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b#:~:text=So%2C%20both%20TensorFlow%20and%20PyTorch,from%20which%20you%20may%20choose.

Forum, W. E. (2015). *Global Risks 2015, 10th Edition.* Geneva: World Economic Forum.

Hadad, Y. (2017, March 16). *30 Amazing Application of Deep Learning*. Retrieved from Yaron Hadad: http://www.yaronhadad.com/deep-learning-most-amazing-applications/

Hamet, P., & Tremblay, J. (2017). Artificial Intelligence in Medicine. *Metabolism*, S36-S40.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv*.

Mirbauer, M., Krabec, M., Křivánek, J., & Šikudová, E. (2021). Survey and Evaluation of Neural 3D Shape Classification. *techrxiv.14447250.v1*.

Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view Convolutional Neural Networks for 3D Shape Recognition. *arXiv*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3D ShapeNets: A Deep Representation for Volumetric Shapes. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1912-1920.

Zanuttigh, P., & Minto, L. (2017). Deep Learning for 3D Shape Classification from Multiple Depth Maps. *IEEE International Conference on Image Processing (ICIP)*, 3615-3619.

Zhang, M., You, H., Kadam, P., Liu, S., & Kuo, C.-C. J. (2020). PointHop: An Explainable Machine Learning Method for Point Cloud Classification. *IEEE Transactions on Multimedia*, 1744-1755.