# 3D Objects Classification With Voxel Grids

Sila Ozturk (ID 2004331) February 10, 2023

*Abstract*—**This paper presents an approach for 3D objects classification using a Voxel Grid VoxNet and Orion. The system utilizes a 3D voxel grid to represent 3D objects and a convolutional neural network (CNN) for feature extraction and classification. The system is evaluated on ModelNet-10 with comparison both VoxNet and Orion without orientation. It also represents the methods effect on training and performance of the network. It shows how they effects with combinations of methods to balance everything with higher accuracy, low time consuming and avoiding over-fitting and under fitting.**

*Index Terms*—**Voxel grid, Optimization, Neural Networks, Convolutional Neural Networks.**

## I. Introduction

Deep learning for 3D data is becoming a more and more important part of machine learning and understanding. With the recent widespread of immersive technologies 3D data formats (e.g. meshes and point clouds) are becoming every day more important. The real world is three dimensional. Compared to 2D visual information, currently very little 3D data is being generated.
In thsi poject, I present my work on classifying 3D CAD models bench-marked on a standard 3D CAD model dataset called the Princeton ModelNet10 with voxel grid neural networks approach. Voxel grid based neural networks are a type of 3D deep learning architecture that operate on volumetric data represented as a 3D grid of voxels. These networks can be used for a variety of tasks, including object recognition, segmentation, and registration, by leveraging the 3D structure of the data.

I've used voxel grid approach for 3D object classification as in the referenced article "Orientation-boosted Voxel Nets for 3D Object Recognition". Basically It has been created with convolutional neural networks (CNN) based architecture that takes a 3D voxelized input. The algorithm is based on the idea of voxelizing the input in order to better detect objects in the 3D space.

The model that I have been developed which fed both VoxNet and Orion ideas by using more hidden layer as like Orion but additionally doing some augmentation and pooling. As Orion It has four Convolutional Neural Networks and two fully connected neural networks but each layer has pooling and data has been augmented to increase data size.

Department of Information Engineering, University of Padova, email: {sila.ozturk}@studenti.unipd.it

## II. Related Work

- **Fusion-Net:** Fusion-Net is a method based on Convolutional Neural Networks, which uses both voxel and pixel representations for training relatively weak classifiers.
- **ORION:** ORION is a type of 3D deep learning architecture designed for object recognition in 3D scenes. The architecture is based on voxel grid based neural networks and utilizes additional information about the orientation of objects in the scene to improve recognition performance. This is achieved by using orientation information in the network's prediction and by introducing an orientation prediction branch to the network architecture. ORION have been shown to outperform other state-of-the-art 3D object recognition methods on benchmark data-sets.
- **VoxNet:** VoxNet is a 3D convolutional neural network (CNN) architecture designed for processing volumetric data, such as 3D scans of objects or environments.The architecture takes advantage of the 3D structure of the data to learn features that capture the geometric properties of the objects being analyzed. The original point cloud discretizes into input data. Points are measured in the neighborhood distance for grouping into various clusters. Every voxel is expressed commonly as 0 for the presence or 1 for the absence in points in the space represented. It has been used for a variety of tasks, including object recognition, segmentation, and registration.
- **Point-Net:** Point-Net is a deep learning architecture designed for processing point cloud data, which is a representation of 3D shapes as a collection of discrete points in space. Point-Net is designed to handle unordered and unstructured point cloud data and operates directly on the point cloud without any prepossessing to voxelize or project the data onto a grid. The architecture uses multi-layer perceptrons to process individual points and a symmetric function to aggregate information across all points in the cloud, allowing it to learn and process the global structure of the shape represented by the point cloud. A raw unranked point cloud represents the 3D data. These methods usually analyze the neighborhood from every point with a given radius, to extract features.

## III. Processing Pipeline

**Remark III.1. Batch Normalization:** Batch normalization is a technique for improving the speed, performance, and stability of neural networks.

Basically, It normalizes the inputs of each layer in a network before activation function, and allows the network to learn faster and reduce the chances of over-fitting. It also helps to reduce the internal co-variate shift, which is the phenomenon where the distribution of the input to a layer changes during training, making it difficult for the network to learn.

By increasing number of layers as like Orion model, at the first It created a mass. For getting close to ORION, 2 more CNN layers has been added, unfortunately without having batch normalization, the network worked inaccurate. So for each CNN layer also batch normalization has been added.
Batch Normalization can be applied to any set of activation's in the network. Here, I focus on transforms that consist of an affine transformation followed by an element-wise non linearity:

$z = g(Wu + b)$

Since we normalize Wu+b, thus, $z = g(Wu + b)$ is replaced with $z = g(BN(Wu))$ where W and b are learned parameters of the model, and g(·) is the non-linearity such as Re-LU where the BN transform is applied independently to each dimension of x = Wu, with a separate pair of learned parameters $\gamma$ (k) , $\beta$(k) per dimension

In traditional deep networks, too-high learning rate may result in the gradients that explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issues.

Batch Normalization also makes training more resilient to the parameter scale. Normally, large learning rates may increase the scale of layer parameters, which then amplify the gradient during back-propagation and lead to the model explosion. However, with Batch Normalization, back-propagation through a layer is unaffected by the scale of its parameters. Indeed, for a scalar a,

$BN(Wu) = BN((aW)u)$ and

we can show that

$\frac{\partial BN((aW)u)}{\partial u} = \frac{\partial BN(Wu)}{\partial u}$

$\frac{\partial BN((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W}$

It has been tried both before and after activation function because, even it is suggested before activation function, some prove show that better after activation function according to some experiment but in my case results shows it must be implemented before activation.

**Remark III.2. Drop Out:** Dropout is a regularization technique for reducing over-fitting in neural networks. It works by randomly dropping out some nodes in the network during training. The idea behind dropout is that by randomly dropping out different nodes during each iteration of the training process, the network is forced to learn multiple redundant representations of the data, rather than relying too heavily on any one node. This helps to prevent over-fitting by making the network less sensitive to the specific weights of individual nodes and more robust to changes in the input data.

For each CNN Layer, the same ratio of dropout did not working well so. For each CNN layer those ratios has been added : 0.2, 0.3, 0.4, 0.6 according to referenced article, I've change ratios, I increased and decreased but as I see nothing changed much, those ratios are already fit well.

**Remark III.3. Average Pooling:** Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively.

First I tried to used Max Pooling but it didn't fit with my model but It was working quite faster and I decided to try average pooling and It worked in accurate way and also fast. After adding avg pooling to each CNN layer, model worked 2 times faster and with 89% accuracy. It helped me to increase batch size from 256 to 512 even 1024 but as my observations, there wasn't much differences using batch size between 512 and 1024 .

Before using pooling, when I tried to train model I was not able to increase batch size because system kept crash because of size of data, so for the system hardware that I have to use, I needed solution which can make me increase batch size. Increasing the performance with pooling I was able to increase number of epoch-es. So in that way both increasing epoch-es and batch size I gather more ac-curated and safe results. Not only performance of model has been increased but also it accuracy has been increased.

At first, last layer pooling was max layer there was 1% difference. After discovering it is working faster than before I reduce learning rate and increase epoch but with 0.1 it is already better I realize that I'm just doing redundant epoch-es with slower learning. And because of batch normalization 0.1 it good enough for learning rate.

According the article, it has been mentioned only one max pooling after fourth CNN layer, but with that I couldn't manage to get accurate results, and I put it each for CNN layer after dropping out.

**Remark III.4. Activation Function:** I've tried both Re-LU $RectifiedLinearUnits$ and leaky Re-LU and actually both worked quite well almost the same accuracy but I rather to use Leaky Re-LU. While Re-Lu function is calculating The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as

$f(x) = max(0, x)f(x) = max(0, x)$

The Leaky Re-LU is one of the most well known. It is the same as Re-LU for positive numbers. But instead of being 0 for all negative values, it has a constant slope less than 1. That slope is a parameter the user sets when building the model, and it is frequently called $\alpha$.
For example, if the user sets $\alpha$=0.3$\alpha$=0.3, the activation function is

$f(x) = max(0.3 * x, x)$.

I've used $\alpha$ for negative slope to control the angle of the negative slope with its default value in PyTorch as 1e-2

**Remark III.5. Loss Function:** For the Loss function, The negative log likelihood loss was used for VoxNet I've tried also for 4CNN layered VoxNet but I've changed it with Cross Entropy Loss function they both worked the same accuracy since One of the differences of the implementation of CrossEntrypyLoss implicitly applies a soft-max activation followed by a log transformation but NLLLoss does not. This criterion combines LogSoftmax and NLLLoss in one single class

Since Cross Entropy is shown as mathematically as below where p is probability of ground truth and q is the probability of predict classes: $H(p, q) = \sum_i p_i \log q_i$

$H(p, q)$ becomes:

$H(p, softmax(output))$.

The lower the bit higher the approximation of f(x). When I use the cross-entropy loss function, the idea of reducing the bit using the optimisation process.

**Remark III.6. Optimizer:** A neural network optimizer is an algorithm used to adjust the weights of a neural network in order to minimize the loss function. Common neural network optimizer include stochastic gradient descent, Adam, and RMSprop.

SGD has been tried as optimizer, it didn't work well without learning rate scheduler. So for the better result, SGD has been used with learning scheduler. This could improved the overall accuracy and generalization of the 3D CNN.

The scheduler can be used to adjust the learning rate, the number of layers, the number of neurons, the activation function, and the optimization algorithm. This allows the network to dynamically adjust its behavior to better learn the data set. This allows the network to more quickly and accurately learn the data set, as the scheduler can adapt to the data set as it changes. Additionally, the scheduler can be used to adjust the networkâ€™s parameters in order to better generalize to unseen data. This allows the network to maintain its accuracy even when presented with new data.
So far, for my experience and since it is with CNN I've tried Adam optimizer but SGD with learning scheduler work better than Adam. So I let it with SGD as an optimizer.

## IV. Data Processing

A voxel is a volumetric pixel, or a three-dimensional pixel. It represents a value on a three-dimensional grid, and is used in many fields such as medical imaging and 3D printing.
It is similar to a pixel in a 2D image, but instead of having a single value associated with it, it has three values associated with it, representing the x, y, and z coordinates of the point in 3D space. A voxel grid is a 3D data structure composed of voxels that are arranged in a regular grid pattern. Voxel grids are used to represent objects in 3D space, and are often used to store and manipulate 3D data. Each meshes as fig.1 :

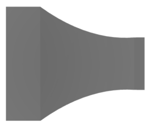

Fig. 1: The Mesh

then each mesh converted to voxel grid as fig.2:

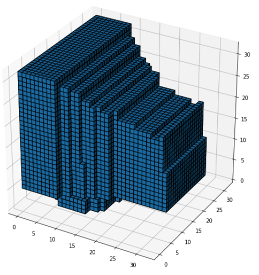

Fig. 2: The Voxel

The objects has been loaded with Trimesh library then they're voxelized then revoxelized to transform each object in the same size with padding. In that way those all shapes were considered in the same size. Whole CAD object converted to mesh then voxel grid with voxelized and revoxelized with 32x32x32. In that way I had whole data-set in voxel gird with size 32x32x32 arrays. While creating data-set, to increase the data set size, data augmentation used and each object flipped and convert to tensor for loading training set and test set.

## V. Method

| | Conv1 | Pool1 | Conv2 | Pool2 |
|---|---|---|---|---|
| # of filters | 32 | | 64 | |
| kernel size | 3 | 2 | 3 | 2 |
| stride | 2 | 2 | 1 | 2 |
| padding | 1 | 0 | 1 | 0 |
| dropout ratio | 0.2 | | 0.3 | |
| batch normalization | ✓ | | ✓ | |

| | Conv3 | Pool3 | Conv4 | Pool4 |
|---|---|---|---|---|
| # of filters | 128 | | 256 | |
| kernel size | 3 | 2 | 3 | 2 |
| stride | 1 | 2 | 1 | 2 |
| padding | 1 | 0 | 1 | 0 |
| dropout ratio | 0.4 | | 0.6 | |
| batch normalization | ✓ | | ✓ | |

| | fc1 | fc2 |
|---|---|---|
| # of filters | 128 | 10 |
| dropout ratio | 0.4 | - |
| batch normalization | x | x |

Data Augmentation has been done for the model to increase data set. I've decided to add two more CNN layers as in reverences article but since training becoming a mass I had to add batch normalization for each CNN layers. The filter sizes are the same as the table above. Two fully connected Layers are following immediately after CNN layers.
Each CNN layers has padding and Pooling. Since Max Pooling didn't?t work good with my model, I decided to use average pooling. For each CNN layer, drop out method has been implemented as it is written table above. For the activation function Re-Lu and Leaky Re-Lu they both worked as the same accuracy but I've decided to use Leaky Re-Lu.

## VI. Results

As an result I can easily compare the both VoxNet and my extended version which based on ORION. Here you can see the architecture of VoxNet, which has two CNN layers and two Fully Connected Layers and max pooling after second CNN layer. The architecture can be seen in Fig.4

The results show that, even VoxNet is already working quite well without any extra method that I used, but it was much 3 times slower than my method. At the end of the training and test, for better comparison for both models, 512 batch size and 0.1 learning rate 60 epoch-es has been used. I started with those parameters for VoxNet, batch size was 128, and epoch-es

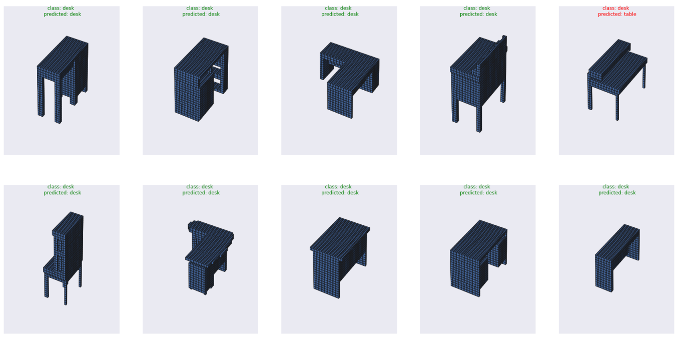

Fig. 3: classification Results

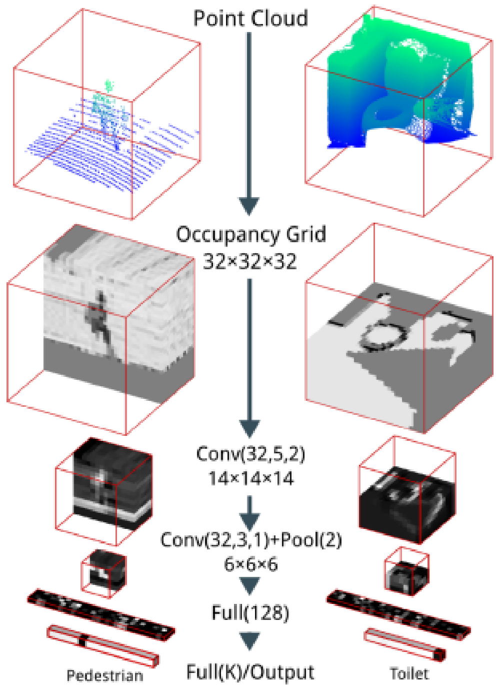

Fig. 4: The VoxNet Architecture

was 20.Accuracy was less than 85% before increase epoch-es and batch size. After changing those parameters result became as below.
Result of VoxNet

```
time: 8134.31 seconds
time: 135.57 minutes
Avg Time Per Epoch 135.57 seconds
Best Val Acc: 89.10%
```

### A. *Experiences*

As result of accuracy and time of training for extended VoxNet or implementation of Orion is:

- **Batch Size 512 With Augmentation & Dropout and 60 epoch-es**

```
Time: 2406.29 seconds
Time: 40.10 minutes
```

```
Avg Time Per Epoch 40.10 seconds
Best Val Acc: 90.86%
```

- **Batch Size 512 with augmentation without dropout**

```
epoch: 20/ 20
train-loss: 0.10,
train-acc: 96.65%
valid-loss: 0.34
valid-acc: 88.44%

Time: 900.47 seconds
Time: 15.01 minutes
Avg Time Per Epoch 45.02 seconds
Best Val Loss: 0.27
Best Val Acc: 88.99%
```

I can say that from train accuracy and validation accuracy, without drop out deference is higher that other results with drop out. For this balance and avoiding to model to learn training set almost 100% accuracy drop out must be use.

- **Batch Size 512 and without augmentation & dropout**

```
epoch: 20/ 20
train-loss: 0.06,
train-acc: 98.66%
valid-loss: 0.29,
valid-acc: 89.76%
time: 45.52 seconds

Total Time: 912.29 seconds
Total Time: 15.20 minutes
Avg Time Per Epoch 45.61 seconds
Best Val Loss: 0.21
Best Val Acc: 90.09
```

With this experience above, still without drop out training accuracy too high and far away from reality even validation accuracy is high unfortunately, model is tend to over-fit.

- **Batch Size 128 with dropout without augmentation**

```
epoch: 20/ 20
train-loss: 0.28,
train-acc: 90.78%
valid-loss: 0.38,
valid-acc: 86.12%
time: 88.64 seconds

Time: 1816.49 seconds
Time: 30.27 minutes
Avg Time Per Epoch 90.82 seconds
Best Val Loss: 40.60%
Best Val Acc: 86.45%
```

As we see experience above, with drop out, we avoid to have too high accuracy ratio in that way the result with drop out is more realistic.

It is quite obvious from the results, even they both VoxNet and my extended version of VoxNet work with close accuracy to each other, with extended version works 3 times faster. So even higher batch size, the model can handle with it having any crashing issue.The number of epoch-es also increased for extended version, accuracy was not changing so much.
I've tried batch size with 128, 256, 512 and 1024. It seems that with 512 has worked well enough as regards 128 and 256. And there wouldn't be any performance differences between 512 and 1024
With lower batch size (128) and without augmentation while training accuracy around 98% test accuracy is around 86% . For training data it is not so much realistic and differences between those to set is quite high. So for lower batch sizes to increase accuracy it is better to use augmentation for creating more various of data.
It is so clear to see from experiences, batch normalization after implementing ORION approach to VoxNet,it has huge importance for accuracy. For the time of each epoch and ending training, from the experiences, It is easy to see average pooling has quite important to have faster training and of-course better performance. With this sampling, It let me to use more batch size and increase the number of epoch-es.
You can see the confusion matrix-es fig.5 and fig.6 and Loss and accuracy histories fig.7 and fig.8 both VoxNet and extended VoxNet by me are like as below figure:

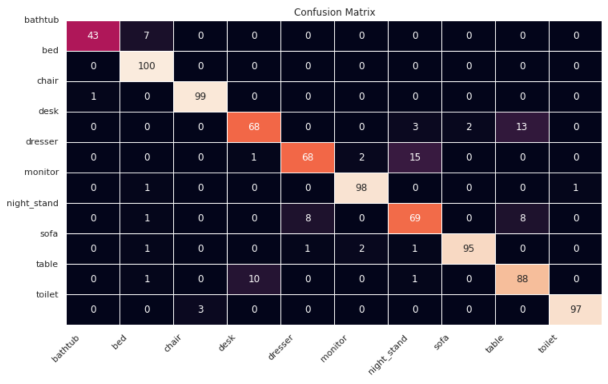

Fig. 5: The Confusion Matrix of Extended VoxNet

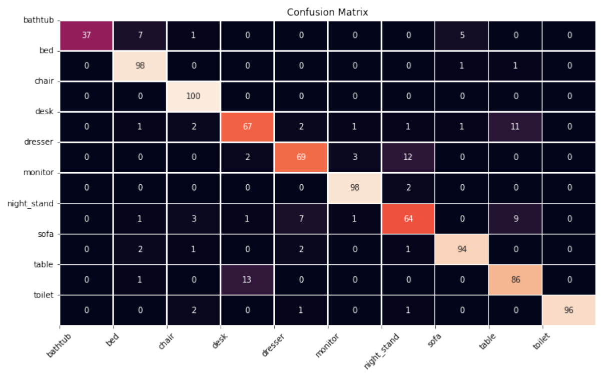

Fig. 6: The Confusion Matrix of VoxNet

I can easily say that both models are quite well about matching classes of object.

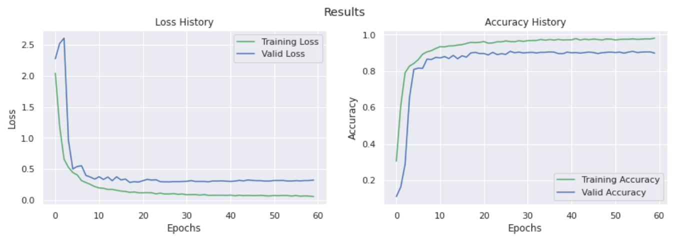

Fig. 7: The Accuracy and Loss History of Extended VoxNet

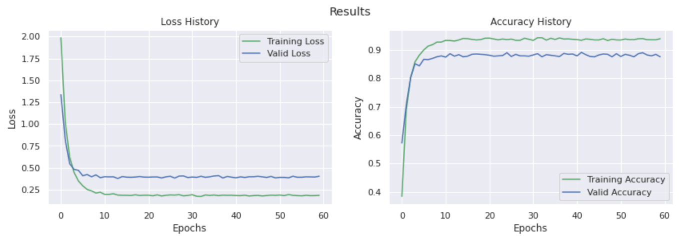

Fig. 8: The Accuracy and Loss History of VoxNet

### B. *Fail Experiences*

I've tried couple of recommendations which I read on internet but as I see they were not exactly suitable for my model.

One of those is using batch normalization after activation function which is not written in its article exactly in that suggesting here on: https://stackoverflow.com/questions/47143521/where-to-apply-batch-normalization-on-standard-cnns, according to comment here:

*"The original batch-norm paper prescribes using the batch-norm before Re-LU activation. But there is evidence that it's probably better to use batch norm after the activation."*

It supposed to work better but for my model normal usage is quite successful so changing place of batch normalization has just created a mass.
The other one is, as I see the place of pooling as immediately after CNN layer almost all examples I saw, but using it after drop out it has already worked quite well.

## VII. Concluding Remarks

Basically we can easily say, increasing batch size and epoch-es helps to increase accuracy. By increasing number of layers needs to be handled with other methods like batch normalization otherwise directly increasing them creates mass. For increasing performance, the approach average pooling worked quite well by decreasing size of tensors, it helps to get faster results and choosing right one according to the model is quite important otherwise even it is fast but won't be have good enough accuracy then it will become useless.
For better controlled predictions and avoid the over fitting and bigger difference between ac-curacies of train and test data, dropout method must be used. Augmentation is worked well with small batch sizes so with the under conditions which we cannot use bigger batch size, it is better to increase various of data by using augmentation.

## References

[1] H. Y. D.-L. Z. B. Jiang and P. Yu, "Negative Log Likelihood Ratio Loss for Deep Neural Network Classification," *Machine Learning (cs.LG)*, Apr. 2017.

[2] V. H. R. Zadeh, "Fusionnet: 3d object classification using multiple data representations," *Computer Vision and Pattern Recognition (cs.CV)*, Nov. 2016.

[3] X. T. J. X. Zhirong Wu Shuran Song Aditya Khosla Fisher Yu Linguang Zhang, "3D ShapeNets: A Deep Representation for Volumetric Shapes," *Computer Vision and Pattern Recognition (cs.CV)*, Apr. 2015.

[4] D. M. S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Dec. 2015.

[5] N. S. M. Z. E. A. T. Brox, "Orientation-boosted voxel nets for 3d object recognition," *Computer Vision and Pattern Recognition (cs.CV)*, Oct. 2017.

[6] S. I. C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Machine Learning (cs.LG)*, Mar. 2015.

[1] [2] [3] [4] [5] [6]