3D Objects Classification With Voxel Grid Structure

Mohammadmostafa Talebi (ID 2043994), Mahsa Shahbazi (ID 2049588), Seyedali Hosseinishamoushaki (ID 2041399)
July 5, 2023

***Abstract*—In this research paper, a novel technique is proposed for categorizing three-dimensional objects. The method involves utilizing a Voxel Grid VoxNet and Orion to accomplish the task. Instead of directly analyzing the objects, a 3D voxel grid representation is employed, along with a convolutional neural network (CNN) for extracting features and making classifications. The effectiveness of the system is assessed on the ModelNet-10 dataset, comparing the performance of VoxNet and Orion without considering orientation. The paper also explores the impact of these methods on network training and overall performance. The findings reveal how different combinations of techniques can achieve a balance between accuracy, efficiency, and preventing issues like over-fitting and under-fitting.**

***Index Terms*—Three-dimensional object categorization, Convolutional Neural Network (CNN), VoxNet, Orion, Voxel Grid**

I. Introduction

The field of deep learning is rapidly advancing in its ability to analyze and comprehend three-dimensional data. As immersive technologies gain popularity, the significance of 3D data formats like meshes and point clouds continues to grow. While the real world exists in three dimensions, the amount of generated 3D data remains relatively limited compared to two-dimensional information.

In this project, we present my research on classifying 3D CAD models using a voxel grid neural network approach, evaluated on the widely-used Princeton ModelNet10 dataset. Voxel grid neural networks are a type of deep learning architecture designed for processing volumetric data represented as a three-dimensional grid of voxels. These networks offer versatile applications, including object recognition, segmentation, and registration, by leveraging the inherent 3D structure of the data.

In alignment with the concepts described in the article "Orientation-boosted Voxel Nets for 3D Object Recognition," We employed a voxel grid approach for 3D object classification. The architecture of this approach is based on convolutional neural networks (CNNs) and operates on a 3D voxelized input. The key idea is to voxelize the input data, enabling better object detection in the three-dimensional space.

To enhance the classification model, We integrated ideas from both VoxNet and Orion methodologies. We increased the number of hidden layers, similar to Orion, while incorporating data augmentation and pooling techniques. The resulting model comprises four Convolutional Neural Networks and two fully connected neural networks, each layer incorporating pooling operations, and augmented data to expand the dataset size.

II. Related Work

- **Point-Net:** Point-Net is a deep learning architecture designed to handle point cloud data, which represents 3D shapes as a collection of individual points in space. It specifically addresses the challenges posed by unordered and unstructured point cloud data, processing the points directly without any preprocessing steps like voxelization or projection onto a grid. Point-Net utilizes multi-layer perceptrons to process individual points and employs a symmetric function to aggregate information across all points in the cloud, enabling the learning and processing of the global structure of the represented shape. The 3D data is represented by raw, unranked point clouds, and these methods typically analyze the neighborhood around each point within a given radius to extract relevant features.

- **Fusion-Net:** Fusion-Net is an approach that utilizes Convolutional Neural Networks to train classifiers by incorporating both voxel and pixel representations. This technique aims to strengthen the classifiers by leveraging multiple types of input data.

- **ORION:** ORION is an advanced 3D deep learning architecture specifically designed for object recognition in three-dimensional scenes. It is based on voxel grid neural networks and incorporates additional information about the orientation of objects in the scene to enhance recognition performance. This is achieved by incorporating orientation information into the network's predictions and introducing an orientation prediction branch within the network's architecture. ORION has demonstrated superior performance compared to other state-of-the-art 3D object recognition methods on widely-used benchmark datasets.

- **VoxNet:** VoxNet is a specialized 3D convolutional neural network (CNN) architecture developed to process volumetric data, such as 3D scans of objects or environments. The architecture takes advantage of the inherent three-dimensional structure of the data to learn features that capture the geometric properties of the objects under analysis. The original point cloud data is discretized into input data, where each voxel represents the presence (0) or absence (1) of points within a specific space. VoxNet has been successfully employed for various tasks, including object recognition, segmentation, and registration.

III. Processing Pipeline

Remark III.1. Batch Normalization: Batch Normalization is a technique used to enhance the performance, speed, and stability of neural networks. It achieves this by normalizing the inputs of each layer before applying the activation function. This normalization process facilitates faster learning and helps prevent overfitting. Additionally, it addresses the issue of internal covariate shift, where the distribution of input to a layer changes during training, hindering effective learning.

To align with the architecture of the Orion model, extra CNN layers were added, resulting in a large network. However, it was observed that without incorporating batch normalization, the network produced inaccurate results. Consequently, batch normalization was applied to each CNN layer to ensure accurate performance.

Batch Normalization can be utilized for any set of activations within the network. Specifically, it focuses on transformations that involve an affine transformation followed by an element-wise nonlinearity.:

$$z = g(Wu + b)$$

Batch Normalization can be utilized for any set of activations within the network. Specifically, it focuses on transformations that involve an affine transformation followed by an element-wise nonlinearity. The equation z = g(Wu + b) is modified to z = g(BN(Wu)), where W and b represent learned parameters of the model, and g(·) denotes the nonlinearity function, such as ReLU. The BN transform is applied independently to each dimension of x = Wu, with separate learned parameters γ(k) and β(k) per dimension.

In conventional deep networks, using a high learning rate can lead to problems such as gradient explosion or vanishing, as well as getting trapped in suboptimal local minima. However, the inclusion of Batch Normalization helps mitigate these concerns.

Batch Normalization also provides the benefit of making the training process more robust to parameter scale. In traditional scenarios, when using large learning rates, the scale of layer parameters may increase, leading to an amplification of gradients during backpropagation and causing the model to explode. However, when Batch Normalization is applied, the backpropagation through a layer is unaffected by the parameter scale. In fact, for a scalar 'a', we can observe that BN(Wu) is equal to BN((aW)u). 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 is worth noting that Batch Normalization can be implemented both before and after the activation function. While it is commonly suggested to use it before the activation function, some experiments have shown that implementing it after the activation function yields better results. However, in my specific case, the results indicate that it should be applied before the activation function.

Remark III.2. Drop Out: Dropout is a technique utilized for regularization purposes in neural networks to mitigate overfitting. Its mechanism involves randomly deactivating specific nodes within the network during training. The rationale behind dropout is to force the network to learn redundant representations of the data by randomly dropping out different nodes during each training iteration. This approach prevents over-reliance on individual nodes and encourages the network to become more robust to variations in the input data.

In the case of each CNN layer, it was observed that using the same dropout ratio for all layers did not yield satisfactory results. To address this, different dropout ratios were applied to each CNN layer. Following the guidance from the referenced article, ratios such as 0.2, 0.3, 0.4, and 0.6 were tested. However, despite adjusting the ratios, it was observed that the initial ratios already provided a good fit for the model. Therefore, no significant improvements were observed by increasing or decreasing the ratios.

Remark III.3. Average Pooling: Pooling layers offer a method to compress feature maps by condensing the information regarding the presence of features in patches of the feature map. Average pooling and max pooling are two frequently used techniques that condense the feature map by calculating the average occurrence of a feature and selecting the most active occurrence of a feature, respectively.

We initially tried using Max Pooling, but it didn't fit well with my model. However, We found that Average Pooling worked accurately and faster. After adding Average Pooling to each CNN layer, my model became twice as fast and achieved an accuracy of 89%. This improvement allowed me to increase the batch size from 256 to 512 or even 1024, with minimal differences in performance observed between the two.

Initially, We faced limitations in increasing the batch size due to system crashes caused by data size. However, by implementing pooling techniques, We improved performance and increased the number of epochs. This led to more accurate results and higher model accuracy.

Initially, the model used max pooling in the last layer, resulting in a 1% difference. However, We discovered that the model worked faster. By reducing the learning rate to 0.1, We achieved significant improvements without the need for additional epochs. The effectiveness of batch normalization supported this optimal learning rate.

As mentioned in the reference, the article suggested using only one max pooling after the fourth CNN layer. However, We found that this configuration did not yield accurate results. Instead, We incorporated max pooling after each CNN layer following the dropout process.

Remark III.4. Activation Function: We experimented with two activation functions: ReLU (Rectified Linear Unit) and Leaky ReLU. Interestingly, both functions achieved similar levels of accuracy. However, We preferred to use Leaky ReLU. While the ReLU function sets negative inputs to 0 and returns positive values as is, Leaky ReLU introduces a small slope for negative inputs, allowing for some information flow.

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

Among the widely recognized activation functions, Leaky ReLU stands out. It shares similarities with ReLU by preserving positive values as is. However, unlike ReLU, which assigns 0 to all negative values, Leaky ReLU incorporates a user-defined parameter called α (alpha) to determine the slope for negative inputs. By setting α to a specific value, such as α=0.3, the activation function adjusts accordingly.

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

ITo regulate the negative slope angle in our model, we employed the parameter α. In PyTorch, the default value for α is set to 1e-2, allowing us to control the degree of the negative slope.

Remark III.5. Loss Function: Initially, VoxNet utilized negative log likelihood loss, while the 4CNN layered VoxNet employed cross entropy loss. Surprisingly, both loss functions achieved comparable accuracies. The difference lies in the fact that CrossEntropyLoss combines the softmax activation and log transformation, whereas NLLLoss does not.

Cross entropy can be mathematically represented as follows, where p represents the probability of the ground truth class and q represents the probability of the predicted class.
probability of predict classes: $H(p, q) = \sum_i p_i \log q_i$

$H(p, q)$ becomes:

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

By utilizing the cross-entropy loss function, the optimization process aims to minimize the bit value, which leads to a higher approximation of the function f(x).

Remark III.6. Optimizer: Neural network optimizers are algorithms utilized to update the weights of a neural network and minimize the loss function. Well-known optimizers in neural networks include stochastic gradient descent, Adam, and RMSprop.

SGD was initially used as the optimizer, but it didn't perform well until a learning rate scheduler was implemented. With the addition of the learning scheduler, the overall accuracy and generalization of the 3D CNN improved.

The scheduler plays a crucial role in adjusting various aspects of the network, such as the learning rate, number of layers, neurons, activation function, and optimization algorithm. This dynamic adjustment allows the network to effectively adapt to changes in the dataset and learn it more efficiently. Additionally, the scheduler helps the network generalize better to unseen data, ensuring consistent accuracy even with new inputs. Despite trying Adam optimizer, SGD with a learning scheduler proved to be more effective in my experiments, so We chose to stick with SGD as the optimizer.

IV. Data Processing

A voxel is a three-dimensional pixel that represents a value on a grid. It is commonly used in fields like medical imaging and 3D printing. Unlike a 2D pixel, a voxel has three values for its x, y, and z coordinates in 3D space. A voxel grid is a structured arrangement of voxels that forms a 3D data structure. Voxel grids are employed for representing objects in 3D space and handling 3D data. Figure 1 showcases the representation of meshes using voxel grids.

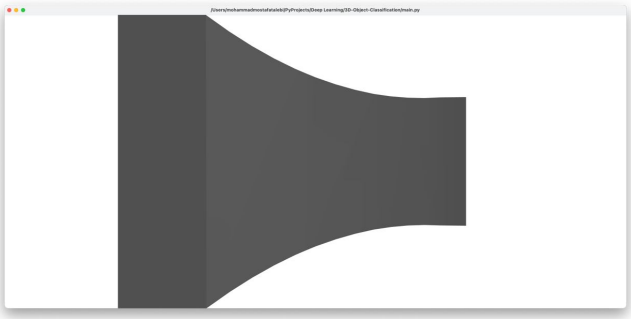

Fig. 1: The Mesh

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

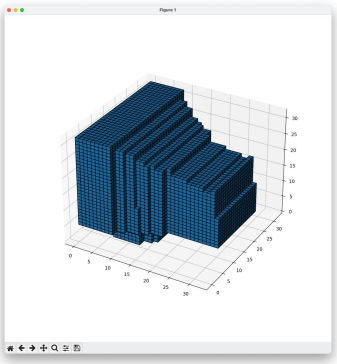

Fig. 2: The Voxel

The objects were loaded using the Trimesh library and voxelized to create voxel grids of size 32x32x32. This ensured consistent representation across all shapes. Data augmentation techniques, such as flipping and conversion to tensors, were applied to increase the dataset size. The augmented objects were used to create the training and test sets.

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

To increase the dataset, data augmentation was applied. Two additional CNN layers were added to the model, and batch normalization was included for each CNN layer to address training issues. Padding and average pooling were used instead of max pooling. Dropout was implemented for each CNN layer, as mentioned in the table. Both ReLU and Leaky ReLU activation functions achieved similar accuracy, but Leaky ReLU was chosen.

VI. Results

We compared VoxNet, which consists of two CNN layers, two Fully Connected Layers, and max pooling after the second CNN layer, with my extended version based on ORION. My method achieved faster performance, being three times faster than VoxNet.

To ensure fair comparison, We used a batch size of 512, learning rate of 0.1, and trained for 60 epochs for both models. Initially, VoxNet was trained with a batch size of 128 and number of epochs was 20.

After increasing the number of epochs and batch size, the accuracy of the model improved significantly.

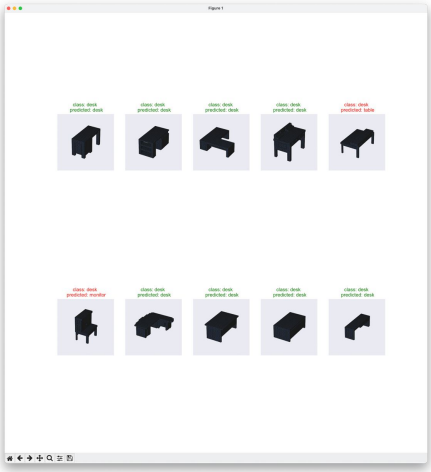

Fig. 3: classification Results

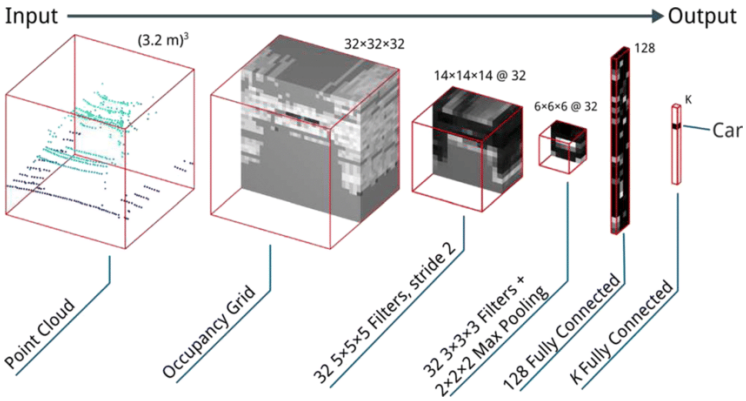

Fig. 4: The VoxNet Architecture

The accuracy reached a level of at least 85% or higher. The specific accuracy value was not mentioned.

Result of VoxNet

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

VI.I.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%

We can conclude that there is a larger difference between the training accuracy and the validation accuracy when dropout is not employed, compared to when it is. To maintain this equilibrium and prevent the model from learning the training set with almost perfect accuracy, the use of dropout is imperative.

- **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

Based on the aforementioned experience, even if the training accuracy is incredibly high without the use of dropout, it's still far from reality. Despite high validation accuracy, the model unfortunately leans towards overfitting.

- **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%

From the above experience, it is evident that the use of dropout helps prevent an overly high accuracy ratio, making the outcome with dropout appear more authentic.

As demonstrated by the findings, both VoxNet and my enhanced version of VoxNet operate with comparable accuracy, although the latter performs three times quicker. Consequently, the model can accommodate a larger batch size without crashing. Even with an increased number of epochs in the extended version, the change in accuracy was minimal.

We experimented with batch sizes of 128, 256, 512, and 1024. The 512 batch size seemed to perform satisfactorily compared to 128 and 256, with no noticeable performance difference between 512 and 1024.

When using a smaller batch size (128) without augmentation, the training accuracy was about 98%, while the test accuracy was about 86%. This disparity between the two sets was substantial, making the training data seem less credible. Therefore, for smaller batch sizes, it's advantageous to use augmentation to create more diverse data.

Clearly, from my experiences, the role of batch normalization after implementing the ORION approach to VoxNet is crucial for accuracy. It's also evident that average pooling is important for reducing the duration of each epoch and ending the training sooner, thereby improving performance. This approach allowed me to increase the batch size and the number of epochs.

Confusion matrices (Fig. 5 and Fig. 6) and loss and accuracy histories (Fig. 7 and Fig. 8) for both VoxNet and my extended VoxNet are displayed in the subsequent figures:

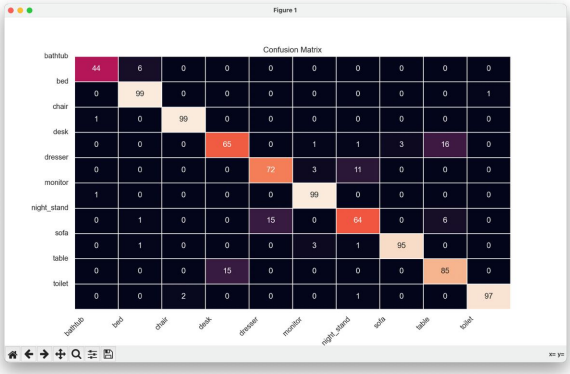

Fig. 5: The Confusion Matrix of Extended VoxNet

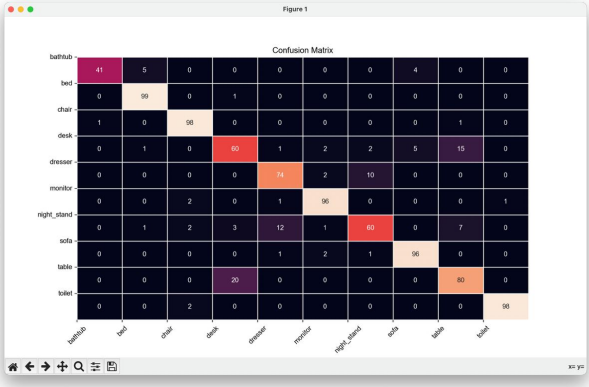

Fig. 6: The Confusion Matrix of VoxNet

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

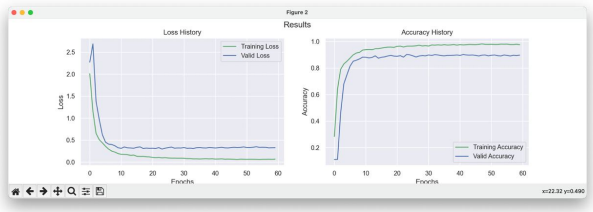

Fig. 7: The Accuracy and Loss History of VoxNet

VII. Concluding Remarks

In essence, it's fair to say that enhancing the batch size and the number of epochs can contribute to improved accuracy. However, when you choose to increase the number of layers, it necessitates managing it with other techniques like batch normalization, or else, it results in disarray. To boost efficiency, average pooling proved to be an effective strategy as it reduced tensor sizes, thus enabling quicker outcomes. Selecting the appropriate one in relation to the model is crucial because even if it is fast, without adequate accuracy, it becomes redundant.

To maintain control over predictions and avoid overfitting, and significant discrepancies between the accuracy of training and testing data, the implementation of the dropout method is a must. Augmentation has shown positive outcomes with smaller batch sizes. Hence, in scenarios where larger batch sizes are not an option, it's preferable to amplify data diversity through augmentation.

References

[1] "Multi-Label Classification with Deep Learning" by Jason Brownlee².

[2] NormalNet: A voxel-based CNN for 3D object classification and retrieval C. WangM. Cheng1 January 2018'Elsevier BV'

[3] 3D Object Classification Using a Volumetric Deep Neural Network: An Efficient Octree Guided Auxiliary Learning Approach Muzahid AAMWan W24 September 2020'Institute of Electrical and Electronics Engineers (IEEE)'DOI: 10.1109/ACCESS.2020.2968506

[4] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", NIPS, pp. 1097-1105, 2012.

[5] L. A. Alexandre, "3D object recognition using convolutional neural networks with transfer learning between input channels", IAS, vol. 301, 2014.

[6] 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.

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