1 Auto-Alignment of the Modelnet40 dataset

Modelnet40 [5] consists of more than 12000 non-aligned objects in 40 classes. We used the method of Sedaghat & Brox [4] to automatically align the objects class by class.

Mesh to Point-Cloud Conversion The auto-alignment method of [4] uses point-cloud representations of objects as input. Thus we converted the 3D mesh grids of Modelnet40 to point-clouds by assigning uniformly distributed points to object faces.

Hidden faces in the mesh grids needed to be removed, as the so called Hierarchical Orientation Histogram (HOH) of [5] mainly relies on the exterior surfaces of the objects. We tackled this issue using the Jacobson’s implementation [6] of the “ambient occlusion” method [3].

We tried to distribute the points roughly with the same density across different faces, regardless of their shape and size, to avoid a bias towards bigger/wider ones. Our basic point-clouds consist of around 50000 points per object, which are then converted to lighter models using the Smooth Signed Distance surface reconstruction method (SSD) [7] as used in [4].

Auto-Alignment We first created a “reference-set” in each class, consisting of a random subset of its objects, with an initial size of 100. This number was then decreased, as the low-quality objects were automatically removed from the reference set, according to [4]. This reference set was then used to align the remaining objects of the class one by one.

For the HOH descriptor, we used 32 and 8 divisions in $\phi$ and $\theta$ dimensions respectively, for the root component. We also used 8 child components with 16 divisions for $\phi$ and 4 for $\theta$ – see [4].

Automatic Assignment of Number of Orientation Classes As pointed out in the main paper, we do not use the same number of orientation classes for all the object categories. We implemented the auto-alignment procedure in a way that this parameter is automatically decided upon for each category: During generation of the reference-set in each class, the alignment procedure was run with 3 different configurations, for which the search space spanned over 360, 180 and 90 degrees of rotations respectively. Each run resulted in an error...
measure representing the overall quality of the models selected as the reference-set, and we
designated respectively 12, 6 and 3 orientation levels to each category, whenever possible.
When none of these worked, e.g. for the ’flower_pot’ class, we assigned 1 orientation class
which is equivalent to discarding the orientation information.

2 Analysis

To analyze the behavior of the orientation-boosted network, we compare it to its correspond-
ing baseline network. We would like to know the differences between corresponding filters
in the two networks. To find this correspondence, we first train a baseline network, without
orientations outputs, for long enough so that it reaches a stable state. Then we use this
trained net to initialize the weights of the ORION network, and continue training with a low
learning rate. This way we can monitor how the learned features change in the transition
from the baseline to the orientation-aware network.

In Figure 1 transition of a single exemplar filter is depicted, and its responses to different
rotations of an input object are illustrated. It turns out that the filter tends to become more
sensitive to the orientation-specific features of the input object. Additionally some parts of
the object, such as the table legs, show stronger response to the filter in the orientation-aware
network.

With such an observation, we tried to analyze the overall behavior of the network for
specific object classes with different orientations. To this end we introduce the “dominant
signal-flow path” of the network. The idea is that, although all the nodes and connections
of the network contribute to the formation of the output, in some cases there may exist a
set of nodes, which have a significantly higher effect in this process for an specific type of
object/orientation. To test this, we take this step-by-step approach: First in a forward pass,
the class, $c$, of the object is found. Then we seek to find the highest contributing node of the
last hidden layer:

$$l^{n-1} = \arg \max_k \{ w^n_{k,c} d^{n-1}_k \}$$ (1)

where $n$ is the number of layers, $d^{n-1}_k$ are the activations of layer $n - 1$, and $w^n_{k,c}$ is the
weight connecting $a^{n-1}_k$ to the $c^{th}$ node of layer $n$. This way we naively assume there is a
significant maximum in the contributions and assign its index to $l^{n-1}$. Later we will see that
this assumption proves to be true in many of our observations. We continue “back-tracing”
the signal, to the previous layers. Extension of (1) to the convolutional layers is straight-
forward, as we are just interested in finding the index of the node/filter in each layer. In
the end, letting $l^n = c$, gives us the vector $l$ with length equal to the number of network
layers, keeping the best contributors’ indices in it. Now to depict the “dominant signal-flow path” for a group of objects, we simply obtain $l$ for every member of the group, and plot
the histogram of the $l$’s as a column. Figure 2(a) shows such an illustration for a specific
class-rotation of the objects. It is clearly visible that for many objects of that group, specific
nodes have been dominant.

In Figure 2(b), the dominant paths of the baseline and ORION networks for some sample
object categories of the Modelnet10 dataset are illustrated. It can be seen that in the baseline
network, the dominant paths among various rotations of a class mostly share a specific set
of nodes. This is mostly visible in the convolutional layers – e.g. see the red boxes. On the
contrary, the dominant paths in the ORION network rarely follow this rule and have more
Figure 1: The picture illustrates the activations of one of the nodes of the first layer, while the network transitions from a baseline network to ORION. The input is always the same object, which has been fed to the network in its possible discretized rotations (columns) at each step (row). We simulated this transition by first training the baseline network and then fine-tuning our orientation-aware architecture on top of the learned weights. To be able to depict the 3D feature maps, we had to cut out values below a specific threshold. It can be seen that the encoded filter detects more orientation-specific aspects of the object, as it moves forward in learning the orientations. In addition, it seems that the filter is becoming more sensitive to a table rather than only a horizontal surface – notice the table legs appearing in the below rows.
Figure 2: (a) shows the “dominant signal-flow path” of the network, for an exemplar object category-orientation. Each column contains the activations of one layer’s nodes. Obviously the columns are of different sizes. Higher intensities show dominant nodes for the specific group of objects. Details of the steps taken to form such an illustration are explained in the text. In (b), rows represent object classes, while in different columns we show rotations of the objects. So each cell is a specific rotation of a specific object category. It can be seen that in the baseline network, many of the rotations of a class, share nodes in their dominant path (e.g. see the red boxes), whereas, in the ORION network the paths are more distributed over all the nodes.
distributed path nodes. We interpret this as one of the results of orientation-boosting, and a helping factor in better classification abilities of the network.

3 Extended Architecture

<table>
<thead>
<tr>
<th></th>
<th>Conv1</th>
<th>Conv2</th>
<th>Conv3</th>
<th>Conv4</th>
<th>Pool4</th>
</tr>
</thead>
<tbody>
<tr>
<td># of filters</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>-</td>
</tr>
<tr>
<td>kernel size</td>
<td>3x3x3</td>
<td>3x3x3</td>
<td>3x3x3</td>
<td>3x3x3</td>
<td>2x2x2</td>
</tr>
<tr>
<td>stride</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>padding</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>dropout ratio</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
<td>-</td>
</tr>
<tr>
<td>batch normalization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

fc1 fc2:classification fc2:orientation

| # of outputs | 128   | 10/40 | variable$^\dagger$ |
| dropout ratio| 0.4   | -     | -                 |
| batch normalization | × | ×     | ×                 |

Table 1: Details of the extended architecture introduced in Tables 1 & 2 of the main article. $^\dagger$ The number of nodes dedicated to the orientation output varies in different experiments.

4 Orientation Estimation Results

Although the orientation estimation was used merely as an auxiliary task, here in Table 2 we report the accuracies of the estimated orientation classes. Note that getting better results on orientation estimation would be possible by emphasizing on this task – e.g. see the detection experiment in the main article.

<table>
<thead>
<tr>
<th></th>
<th>Accuracy %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sydney</td>
</tr>
<tr>
<td>ORION</td>
<td>71.5</td>
</tr>
<tr>
<td>ORION – Extended</td>
<td>70.1</td>
</tr>
</tbody>
</table>

Table 2: Orientation estimation accuracies on different datasets. The extended architecture of the second row, is the one introduced in the main article and detailed in Table 1 of this document.
References


