Eurographics Workshop on 3D Object Retrieval (2016)
A. Ferreira, A. Giachetti, and D. Giorgi (Editors)

SHREC'16 Track
Large-Scale 3D Shape Retrieval from ShapeNet Core55

M. Savva[1*], F. Yu[2*], Hao Su[1*], M. Aono[8], B. Chen[6], D. Cohen-Or[5], W. Deng[7], Hang Su[3], S. Bai[4], X. Bai[4], N. Fish[5], J. Han[7],

E. Kalogerakis[3], E. G. Learned-Miller[3], Y. Li[5,6], M. Liao[4], S. Maji[3], A. Tatsuma[8], Y. Wang[7], N. Zhang[7] and Z. Zhou[4]

[1]Stanford University, USA
[2]Princeton University, USA
[3]University of Massachusetts Amherst, USA
[4]Huazhong University of Science and Technology, China
[5]Tel Aviv University, Israel
[6]Shandong University, China
[7]Beijing University of Posts and Telecommunications, Beijing, China
[8]Toyohashi University of Technology, Japan
[*]Track organizers
shrec2016shapenet@gmail.com

Abstract
With the advent of commodity 3D capturing devices and better 3D modeling tools, 3D shape content is becoming increasingly prevalent. Therefore, the need for shape retrieval algorithms to handle large-scale shape repositories is more and more important. This track aims to provide a benchmark to evaluate large-scale shape retrieval based on the ShapeNet dataset. We use ShapeNet Core55, which provides more than 50 thousands models over 55 common categories in total for training and evaluating several algorithms. Five participating teams have submitted a variety of retrieval methods which were evaluated on several standard information retrieval performance metrics. We find the submitted methods work reasonably well on the track benchmark, but we also see significant space for improvement by future algorithms. We release all the data, results, and evaluation code for the benefit of the community.

Categories and Subject Descriptors: H.3.3 [Computer Graphics]: Information Systems- Information Search and Retrieval

1. Introduction

3D content is becoming increasingly prevalent and important to everyday life, due to the ubiquity of commodity depth sensors and better 3D modeling tools. The increasing availability of 3D models requires scalable and efficient algorithms to manage and analyze them, to facilitate applications such as virtual reality, 3D printing and manufacturing, and robotics among many others. A key research problem is retrieval of relevant 3D models given a query model. The community has been actively working on this task for more than a decade. However, existing algorithms are usually evaluated on datasets with few models. To process the millions of 3D models available on the Internet, algorithms should be both accurate and scalable, which calls for large-scale benchmark datasets to be used in developing and evaluating shape retrieval methods. Thanks to the recent effort in creating ShapeNet [**?**], we can now use a much bigger dataset of 3D models to develop and evaluate new algorithms. In this track, we evaluate the performance of cutting edge deep-learning based 3D shape retrieval methods on ShapeNet Core55, which is a subset of the ShapeNet dataset with more than 50 thousand models in 55 common object categories.

Five participating teams have submitted a variety of retrieval methods. All the methods are based on deep learning models and projection-based inputs. We evaluate the methods based on several standard information retrieval performance metrics. The submission by Hang Su et al. performs the best on aligned models, while the submission by S. Bai et al. performs the best on models with perturbed rotations. Both the data and evaluation code are publicly released for the benefit of the community. As our results show that there is still space for us to improve upon, we hope that our release will catalyze future research into 3D model retrieval.

DOI: 10.2312/3dor.20161092

2. Dataset

The ShapeNet Core55 competition dataset contains a total of 51190 3D models categorized into 55 WordNet [?] synsets (lexical categories belonging to a taxonomy of concepts in the English language). Before using this data for the competition, the models were deduplicated. Furthermore, each model was assigned a sub-synset (sub-category) label which indicates a more refined class for the object (e.g., a model may have a sub-synset of "fighter jet" within the synset of "airplane". There are 204 sub-synset labels across the 55 synsets and these are used to establish a gradated relevance score (beyond just matching vs not matching synset label) in one of the evaluation metrics.

The original models in the evaluation dataset were taken from the ShapeNet corpus which was collected primarily from the Trimble 3D warehouse [?]. To make training and evaluation of learning based methods possible, we established standard training, validation and test splits of the dataset. The proportions chosen were 70% (35765), 10% (5159) and 20% (10266) respectively out of the total 51190 models. The ShapeNet model data was converted to OBJ format with only geometric information, and the model dimensions were normalized to a unit length cube. In addition, since ShapeNet provides consistent upright and front orientation annotation for all models in the ShapeNetCore corpus, all model data was consistently aligned. We call this the "normal" dataset. To establish a more challenging version of the data without the assumption for pre-existing consistent alignments, we generate a "perturbed" version of the model data where each model has been randomly rotated by a uniformly sampled rotation in $SO(3)$. As most deep-learning methods rely on view-based feature computation, the second dataset is a more challenging scenario. All participating methods were evaluated on both datasets.

3. Evaluation

Each split of the dataset (train, val, test) was treated independently as a query and retrieval corpus. Participants were asked to return a ranked list of retrieved models for each model in a given set, where the target models to be retrieved were all models in that set, including the query model itself. Retrieval ranked lists were required to provide at most 1000 results in descending order of similarity (or relevance) to the query model. Although a similarity score is provided for each entry in the submission, it is not used in evaluation. Each participant method was free to choose whether to return fewer relevant retrieval results to obtain better evaluation performance.

The ranked lists were evaluated against the ground truth category (synset) and subcategory (subsynset) annotations. An item in the retrieved lists is positive if it is in the same category with the target model in the retrieval. Otherwise, it is negative. For each entry in the lists, we calculate the precision and recall. The precision at an entry is defined as the percentage of positive items up to this entry. The recall at an entry is defined as the number of positive items up to this entry divided by the smaller number between the total number of objects in the category or maximally allowed retrieved list length (1000 in this competition). Combining the precision and recall at each entry, we calculate the F-score.

At each entry, we also calculate normalized discounted cumulative gain (NDCG). The NDCG metric uses a graded relevance: 3 for perfect category and subcategory match in query and retrieval, 2 for category and subcategory both being same as the category, 1 for correct category and a sibling subcategory, and 0 for no match. Subcategory is only used in NDCG. The simplistic graded relevance we defined using just categories and subcategories is a somewhat limited attempt at capturing a human notion of graded relevance between 3D models. In the near future, the track organizers plan to collect judgments of retrieved 3D model relevance from people in order to establish a more proper relevance for retrieved models. This will allow computation of a more challenging version of the NDCG evaluation metric which will measure the degree to which retrieval methods match human notions of relevance/similarity ordering of the 3D models.

In summary, we can calculate four scores at each entry in a list: precision, recall, F-score and NDCG. In Section 5.5, we show precision-recall plots to compare different submitted methods. Besides using average precision to evaluate the quality of each retrieved list, we also use precision, recall, F1 and NDCG at the end of the retrieval list as a summary metric. These metrics are referred to as P@N, R@N, F1@N and NDCG@N, where the N refers to the total retrieval list length chosen by the method, and is allowed to vary across queries. The scores of the lists in a category are combined by taking the average.

In order to combine the retrieval results of different categories, we use two versions of the above metrics: macro and micro averaged. The macro-averaged version is used to give an unweighted average over the entire dataset. The retrieval scores for all the models are averaged with equal weights. In the micro-averaged version, each query and retrieval results are treated equally across categories, and therefore the results are averaged without reweighting based on category size. This gives a representative performance metric average across categories.

4. Participants

There were five participating groups in this track. Each group submitted results on both the normal and perturbed versions of the data as well as for each of the dataset splits (train, val, test), with one exception of a group that did not submit train results.

- **MVCNN**: Multi-view Convolutional Neural Networks, by H. Su, S. Maji, E. Kalogerakis, E. G. Learned-Miller (referred to as **Su**)
- **GIFT**: A Real-time and Scalable 3D Shape Search Engine, by S.Bai, Z.Zhou, M.Liao, X.Bai (referred to as **Bai**)
- **ViewAggregation**: 3D Model Signatures via Aggregation of Multi-view CNN Features, by Y.Li, N. Fish, D. Cohen-Or and B. Chen (referred to as **Li**)
- **CCMLT**: Channel-wise CNN for Multitask Learning by Triplet, by Y. Wang, N. Zhang, J. Han and W. Deng (referred to as **Wang**)
- **DB-FMCD-FUL-LCDR**: Appearance-based 3D Shape Feature Extraction using Pre-trained Convolutional Neural Networks, by A. Tatsuma and M. Aono (referred to as **Tatsuma**)

5. Methods

In this section we compile the description of methods by each participating group, from the participants' perspective.

5.1. Multi-view Convolutional Neural Networks, by H. Su, S. Maji, E. Kalogerakis, E. G. Learned-Miller

Our method is based on the Multi-view Convolutional Neural Networks (MVCNN) [?]. Our code is available at `http://vis-www.cs.umass.edu/mvcnn/`.

5.1.1. Pre-processing

Our method takes rendered views of 3D shapes as inputs. We use two different camera setups for rendering. For the **1st camera setup**, we assume that the shapes are upright oriented along a consistent axis (*e.g.* z-axis). The non-perturbed shapes provided in the contest satisfy this assumption, as well as most modern online shape repositories. In this case, we create 12 rendered views by placing 12 virtual cameras around the shape every 30°. For the **2nd camera setup**, which is used for the perturbed shapes, we make no assumption about the orientation of the shapes and place 20 virtual cameras at the 20 vertices of an icosahedron enclosing the shape. We generate 4 rendered views from each camera, using 0°, 90°, 180°, 270° in-plane rotations, yielding a total of 80 views.

The 55 categories in ShapeNetCore55 are highly imbalanced. In the training set, the largest category has about 150 times more shapes than the smallest category. The subcategories are even more imbalanced. To perform category balancing, we apply Eq. 1 to the training class distribution $\mathbf{d}$, and randomly sample a training set for each training epoch according to $\mathbf{d}_{balanced}$. t is a parameter that controls the trade-off between macro-averaged and micro-averaged evaluation metrics. We set t to 0.5 for training the 55-category network and 0.2 for the 204-subcategory network.

$$\mathbf{d}_{balanced}(k) = avg(\mathbf{d}) \cdot (\frac{\mathbf{d}(k)}{avg(\mathbf{d})})^t \tag{1}$$

5.1.2. Network Architecture

As shown in Figure 1, each rendered view of a 3D shape is passed through the first part of the network (CNN_1) separately, aggregated at a view-pooling layer using max-pooling, and then passed through the remaining part of the network (CNN_2). All branches of CNN_1 share the same parameters. CNN_1 and CNN_2 together contain 5 convolutional layers ($\text{conv}_{1,\dots,5}$) and 3 fully-connected layers ($\text{fc}_{6,\dots,8}$). For best performance, the view-pooling layer should be placed somewhere between conv_5 and fc_7 [?]. We use fc_7 for all our submitted results.

We initialize the parameters of the network from the VGG-M network [?], and fine-tune the network on ShapeNetCore55.

5.1.3. Retrieval

We train two networks for each camera setup: one for 55-way classification and another for 204-way subcategory classification. For each query, we first predict its label and construct a retrieval set containing all shapes with the same predicted label. Then we extract features for the query and the targets from the output layer of the 204-way subcategory network (*i.e.* the features are the classification probabilities) and re-rank the results according to their L2 distances to the query. The re-ranking step will not influence precision and recall, and is designed mainly for improving NDCG.

In [?], we used the penultimate layer in the network as features together with low-rank Mahalanobis metric learning for dimension reduction. For this contest, we use the output layer as features directly due to time constraints.

5.2. GIFT: A Real-time and Scalable 3D Shape Search Engine, by S.Bai, Z.Zhou, M.Liao, X.Bai

Our method is based on GIFT [?]. It is composed of four components: projection rendering, view feature extraction, multi-view matching and re-ranking (see Figure 2).

5.2.1. Projection rendering

For each 3D shape, we place its centroid at the origin of a unit sphere and normalize its scale by resizing its maximum polar distance to unit length. Then we render the 3D shape into depth images from N_v ($N_v = 64$ in our experiments) viewpoints located uniformly on the surface of a sphere.

5.2.2. View feature extraction

A convolutional neural network (CNN) is used to extract view features. In the training phase, we fine-tune a pre-trained CNN (VGG-S from [?]) using the rendered views. Each depth view is taken as a training exemplar, with the label of its corresponding 3D shape.

Since each shape in the competition dataset has two labels (category and subcategory), the training loss of the CNN consists of two parts. This setting is different from the CNN in GIFT [?], where only one loss function is used. The first part is a vector, whose length is equivalent to the number of subcategories in the training dataset. Each element of the vector denotes the classification probability of the training examplar to the corresponding subcategory. According to the relationship between category and subcategory, we map this vector to a new vector whose length is the number of main categories. This second part describes the probability distribution over the main category. Specifically, for a given main category, we sum the probability of all its subcategories. Lastly, the final training loss is combined via negative log-likelihood criterion.

In the testing phase, we take the activation of the fully-connected layer L_7 as the feature of each view, which we empirically determined to be the most discriminative layer.

5.2.3. Multi-view matching

Given a query shape x_q and a certain shape x_p from the database $\mathcal{X} = \{x_1, x_2, \dots, x_N\}$, we can easily obtain two feature sets $\mathcal{V}(x_q) = \{q_1, q_2, \dots, q_{N_v}\}$ and $\mathcal{V}(x_p) = \{p_1, p_2, \dots, p_{N_v}\}$ respectively using the trained neural network.

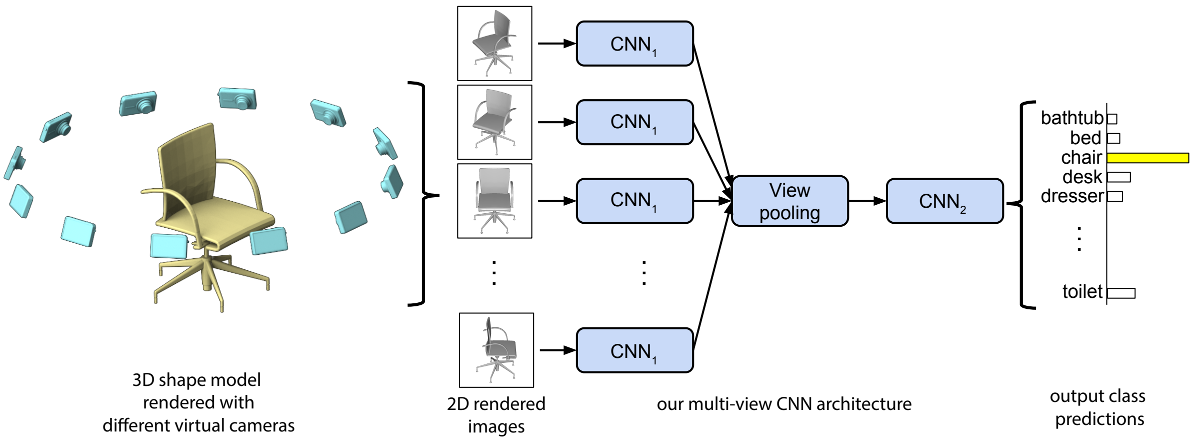

Figure 1: *Overview for **MVCNN Team**. Illustrated using the 1st camera setup for non-perturbed shapes.*

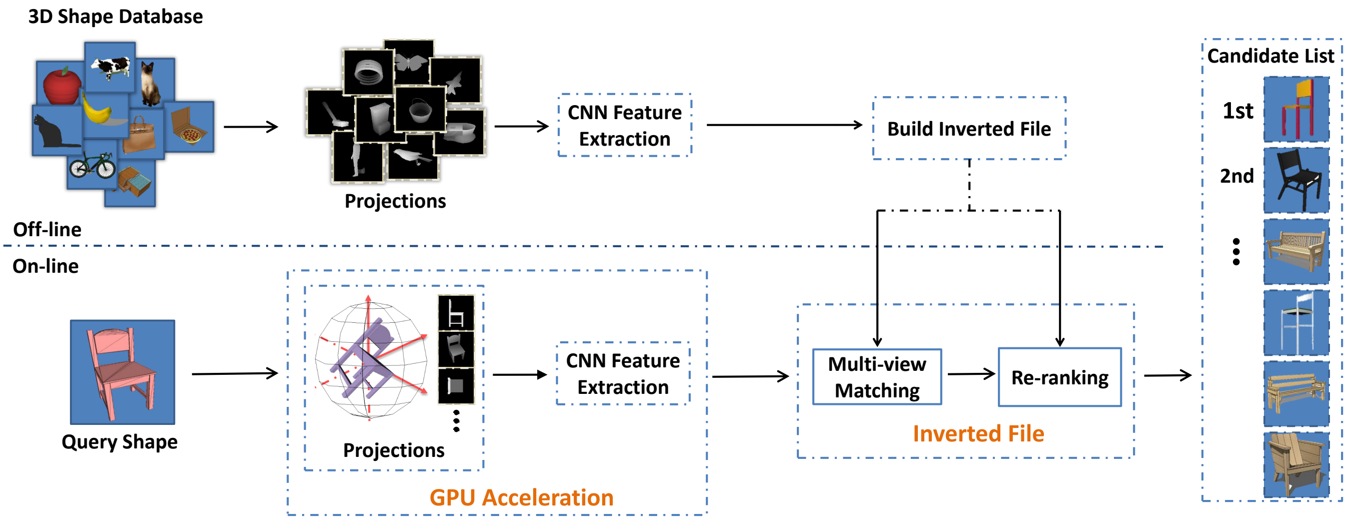

Figure 2: *Overview for **GIFT Team**. Illustration of the overall GIFT pipeline.*

In order to compute the dissimilarity between x_q and x_p, we utilize a robust version of Hausdorff distance, defined as

$$D(x_q, x_p) = \frac{1}{N_v} \sum_{q_i \in \mathcal{V}(x_q)} \min_{p_j \in \mathcal{V}(x_p)} d(q_i, p_j), \tag{2}$$

where $d(\cdot)$ computes the Euclidean distance between two input vectors. Compared with standard Hausdorff distance, this modified version can eliminate the disturbance of isolated views in $\mathcal{V}(x_q)$.

In [?], the calculation of Eq. (2) is further accelerated using an inverted file, thus leading to an approximated Hausdorff distance. The approximated version can shorten the pairwise matching time significantly by sacrificing some retrieval accuracy. Here, we directly use Eq. (2) for better performance.

5.2.4. Re-ranking

Our last component is a context-based re-ranking algorithm.

Our basic idea is that the similarity between two shapes can be more reliably measured by comparing their neighbors using Jaccard similarity. Instead of using a crisp set, we define the neighborhood set of x_q in fuzzy set theory to attach more importance to top-ranked neighbors. Consequently, its k-nearest neighbors set can be described as $\mathcal{N}_k(x_q)$, where each element x_i has membership grade $m(x_i) > 0$. We initialize $m(x_i)$ as $S(x_q, x_i)$, which is the similarity between x_q and x_i. $S(x_q, x_i)$ can be obtained by applying Gaussian kernel to Eq. (2). For those shapes that are not among the top-k ranking list, $m(x_i) = 0$.

For the sake of convenience in comparison, one can easily convert the membership distribution into a vector $F_q \in \mathbb{R}^N$ as $F_q[i] = m(x_i)$. In fuzzy set theory, the Jaccard similarity between $\mathcal{N}(x_q)$ and $\mathcal{N}(x_p)$ can be computed as

$$S'(x_q, x_p) = \frac{\sum_{i=1}^{N} \min(F_q[i], F_p[i])}{\sum_{i=1}^{N} \max(F_q[i], F_p[i])}. \tag{3}$$

S' is used later to produce the refined ranking list for the query x_q.

Since F_q itself is a sparse vector, Eq. (3) can be computed using an inverted index (refer to [?] for a rigorous proof). Moreover, the second-order neighbor can be considered by augmenting F_q as $F_q = \frac{1}{k_1}\sum_{x_i \in N_{k_1}(x_q)} F_i$. These two improvements can improve the matching speed and matching accuracy of Eq. (3).

5.3. 3D Model Signatures via Aggregation of Multi-view CNN Features, by Y. Li, N. Fish, D. Cohen-Or and B. Chen

A 3D model can be rendered into a 2D image from multiple viewpoints, thus a possible signature for it can be obtained by an assembly of multi-view image features [?]. To generate a high quality shape signature, image features must be informative and appropriately discriminative. In recent years, image features extracted from a CNN were shown to be highly successful in image-based recognition tasks, as they are both informative and discriminative. Considering the aforementioned multi-view render-based shape signature paradigm, such advances in image feature extraction can be levereged to boost the performance of shape signatures, for tasks such as shape retrieval.

Following [?], we represent a 3D model signature with CNN features computed on a set of images rendered from the 3D model from multiple viewpoints. We first extract CNN features for each rendered view, with the convolutional layers of a CNN model fine-tuned for classifying each individual rendered view into the category and sub-category of the 3D model, and then aggregate the view features. Finally, we train several fully-connected layers on classification tasks based on the aggregated features.

In this approach, the aggregation of CNN features from different views is key. In [?], the view features are aggregated via max-pooling, such that all views are equally contributive. This approach is therefore oblivious to the manner in which rendering views are chosen. This is an advantage when models within a collection are arbitrarily oriented, but a disadvantage when a consistent alignment is provided. An aggregation method that is consistency-aware is likely to outperform its oblivious counterpart in this instance. Thus, when a consistent shape alignment is given, instead of aggregation via max-pooling, we aggregate multi-view CNN features by concatenating view-specific features in-order. Here, the fully-connected layers can leverage the correspondence that exists between renderings from consistent viewpoints across different shapes, and make decisions based on a more holistic understanding of the various viewpoints. See Figure 3 for a visual representation of the two CNN architectures employed in our approach.

More specifically, we render each model from 12 evenly spaced viewpoints, and fine-tune the VGG-19 network [?] for the convolutional layers that are used for view feature extraction. We use the concatenation of the category and sub-category classification probability vector as the 3D model signature for retrieval. We open source our code at http://yangyanli.github.io/SHREC2016/.

5.4. Channel-wise CNN for Multitask Learning by Triplet, by Y. Wang, N. Zhang, J. Han and W. Deng

We render each 3D model into 36 channels of data by concatenating 36 2D rendered images in sequence. These images are captured from 12 camera positions on the vertices of an icosahedron, for each of 3 orthogonal view up directions. Each channel represents a single pose of the model. Multi-channel data makes it possible to train a feature fusion matrix inside a CNN in contrast to image-based 3D model recognition where features extracted from 2D images of a single model need to be fused again. By avoiding the need to fuse again, the performance is better than single channel data regardless of whether the models are perturbed or not.

A Channel-wise Convolutional Neural Network is used to exploit geometric information in 3D models from this 36 channel data. AlexNet [?] is used as basic parametric model with the layers modified to handle 36 channel data. As images in each channel differ significantly due to varying camera positions, convolutional layers with filters initialized as Xavier filters [?] are trained in a channel-wise fashion to extract discriminant patterns from each channel. The locally connected layer are initialized with the method of MSRA [?] used as the last convolutional layer, adjusting for local patterns. The batch normalization [?] during iterations which is added after every convolution layer is modified with a moving average $Mean_{(t)} = \alpha BatchMean_{(t)} + (1-\alpha)Mean_{(t-1)}$ and $Variance_{(t)} = \alpha BatchVariance_{(t)} + (1-\alpha)Variance_{(t-1)}$.

We perform an operation similar to the z-score operation $\hat{x}^{(k)} = \frac{x^{(k)} - E(x^{(k)})}{\sqrt{Var(x^{(k)})}}$ on the inputs $x^{(k)}$ of the batch normalization layer, then a linear transformation $y^{(k)} = \beta\hat{x}^{(k)} + \gamma$ is applied on $\hat{x}^{(k)}$ where k indicates the number of channels. All operations are performing channel-wise during each iteration t. Denoting by $local\#$ a local connected layer, by $conv\#$ a convolutional layer where the size of the feature map is defined as width×height×channel, by $lrn\#$ a local response normalization layer, by $pool\#$ a pooling layer, by $fc\#$ a fully connected layer, by $relu\#$ a linear unit transformation and by $bn\#$ a moving batch normalization layer, the concatenated structure of the CNN learning and feature fusion in Fig.4 based on channel-wise concatenated data is defined by: $data(225\times225\times36) - conv1(54\times54\times8)\{12\ \mathbf{groups}\} - bn1 - relu2 - lrn1 - pool3 - conv2(27\times27\times64)\{4\ \mathbf{groups}\} - bn2 - relu3 - lrn2 - pool4 - conv3(13\times13\times384) - bn3 - relu4 - conv4(13\times13\times192)\{2\ \mathbf{groups}\} - bn4 - relu5 - local5(13\times13\times128)\{2\ \mathbf{groups}\} - bn5 - relu6 - pool6 - fc3(4096) - fc4(4096) - dropout - fc5(210)$.

The loss $\mathcal{L}_{recog}$ for the CNN is defined as a linear combination of softmax loss and triplet loss for multitask learning which separates different categories and also discriminates on subcategories:

$$\mathcal{L}_{recog} = \alpha\mathcal{L}_{softmax} + \sum_{i=1}^{3}\beta\mathcal{L}_{tri}(s_r, s_p, s_{ni}) \quad (4)$$

$$\mathcal{L}_{tri} = \ln(\max(1, 2 - \frac{||f(x_i) - f(x_k)||_2^2}{||f(x_i) - f(x_j)||_2^2 + m})) \quad (5)$$

where $f(x)$ is the input of the loss layer for sample x and m is the margin(0.01) for the triplet. The proportion of positive and negative pairs in $\mathcal{L}_{tri}$ avoids the gradient vanishing problem in [?]. That is, we avoid the issue of less effective back propagation when models with the same subcategory label are used for training (distance is small between features from the same subcategory in the triplet making the propagated gradient small). Correspondingly, the 36-channel input data generated from a perturbed model makes gradient explosion likely as each channel varies significantly, so we further apply natural logarithms on the loss of a triplet. A triplet

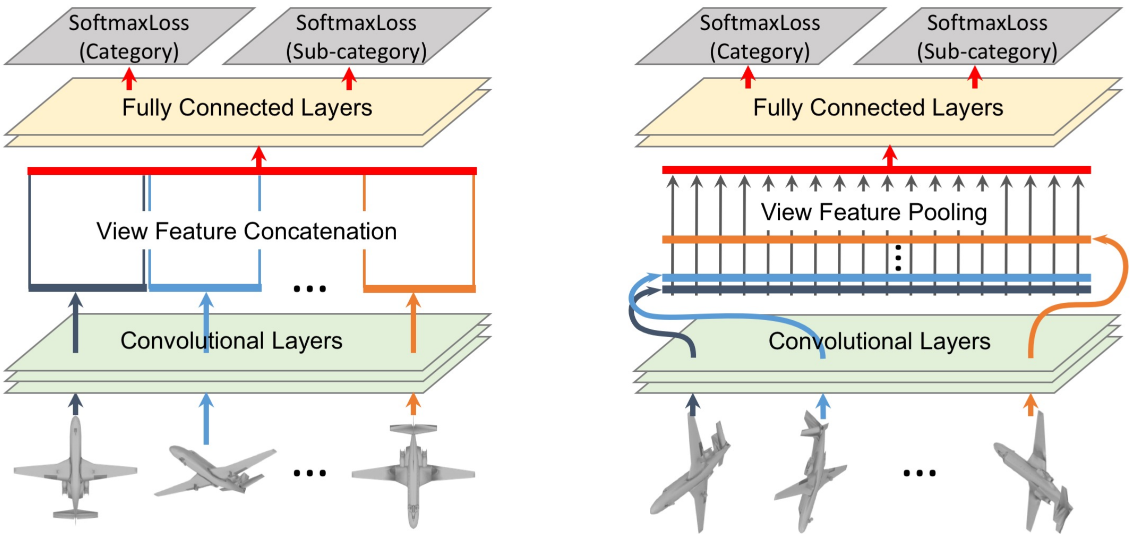

Figure 3: *Overview for **ViewAggregation Team**. CNN architectures for extracting 3D model signatures by aggregating CNN features of multi-view images rendered from 3D models. Multi-view CNN features are aggregated by concatenating per-view features when a consistent 3D model orientation is available (left), or by max-pooling when model orientations are unknown (right).*

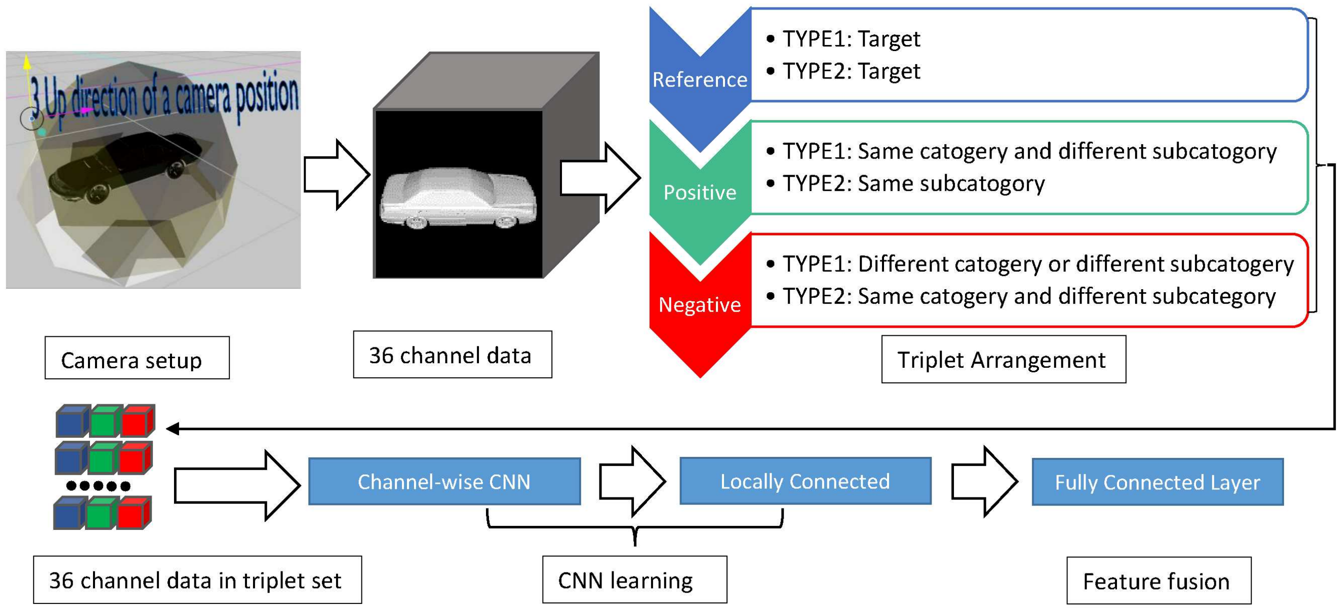
3 Up direction of a camera position
Reference
• TYPE1: Target
• TYPE2: Target
Positive
• TYPE1: Same catogery and different subcatogory
• TYPE2: Same subcatogory
Negative
• TYPE1: Different catogery or different subcatogery
• TYPE2: Same catogery and different subcategory
Camera setup
36 channel data
Triplet Arrangement
Channel-wise CNN
Locally Connected
Fully Connected Layer
36 channel data in triplet set
CNN learning
Feature fusion

Figure 4: *Overview for **CCMLT Team**. Pipeline of data arrangement and CNN learning.*

set composed of 3 triplets is applied for subcategory classification, consisting of 1 reference sample s_r, 1 positive sample s_p and 3 negative samples s_{ni}. As shown in Figure 4, at least 2 negative samples are from a different category. The positive sample is either one in the same subcategory or in the same category. The last negative sample is one in a different category or the same category correspondingly.

5.5. Appearance-based 3D Shape Feature Extraction using Pre-trained Convolutional Neural Networks, by A. Tatsuma and M. Aono

We propose a method that extracts 3D shape features by using a pre-trained Convolutional Neural Network (CNN) [?]. The overview of our approach is illustrated in Figure 5. Our method calculates the Feature Maps Covariance Descriptor (FMCD) [?] on each depth-buffer image rendered for a given 3D model.

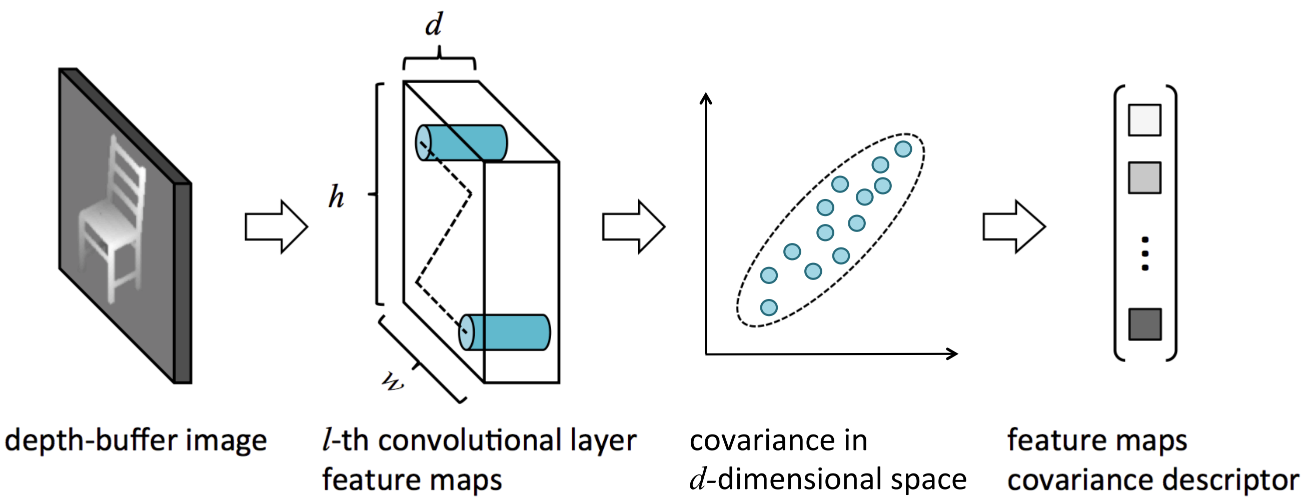
d
h
w
depth-buffer image
l-th convolutional layer feature maps
covariance in d-dimensional space
feature maps covariance descriptor

Figure 5: *Overview for **DB-FMCD-FUL-LCDR Team**. The overview of feature extraction process based on the feature map covariance descriptor.*

As a pre-processing step, to remove the arbitrariness of positioning and scale for the 3D model, we translate the center of the 3D model to the origin and then normalize the size of the 3D model to the unit sphere. In addition, for the perturbed data set, we normalize the rotation of the 3D model by using Point SVD [?].

After the normalization, we render depth-buffer images with 224×224 resolution from 38-vertices on the unit geodesic sphere arranged as to enclose the 3D model.

We obtain an effective feature vector of the 3D model by extracting the FMCD from each depth-buffer image. FMCD comprises the covariance of convolutional layer feature maps on the CNN. We consider the d feature maps of size $w \times h$ outputted from the l-th convolutional layer to be the d dimensional local features of $n = w \times h$ points. Let $F = [\mathbf{f}_1, \dots, \mathbf{f}_n] \in \mathbb{R}^{d \times n}$ denote the set of local features. To obtain a feature vector of a depth-buffer image, we calculate covariance matrix of the local features

$$C = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{f}_i - \mathbf{m})(\mathbf{f}_i - \mathbf{m})^\top,$$

where $\mathbf{m}$ is the mean of the local features.

The covariance matrix lies not on the Euclidean space, but on the Riemannian manifold of symmetric positive semi-define matrices. To solve this problem, we project the covariance matrix onto a point in the Euclidean space using the mapping method proposed by Pennec et al. [?].

The mapping method first projects the covariance matrix onto the Euclidean space that is tangent to the Riemannian manifold at the tangency point P. The projected vector $\mathbf{y}$ of the covariance matrix C is given by

$$\mathbf{y} = \log_P(C) = P^{\frac{1}{2}} \log(P^{-\frac{1}{2}} C P^{-\frac{1}{2}}) P^{\frac{1}{2}},$$

where $\log(\cdot)$ is the matrix logarithm operator. The mapping method extracts the orthonormal coordinates of the projected vector that are given by the following vector operator

$$\mathrm{vec}_P(\mathbf{y}) = \mathrm{vec}_I(P^{-\frac{1}{2}} \mathbf{y} P^{-\frac{1}{2}}),$$

where I is the identity matrix, and the vector operator at identity is defined as

$$\mathrm{vec}_I(\mathbf{y}) = \left[y_{1,1} \sqrt{2} y_{1,2} \sqrt{2} y_{1,3} \dots y_{2,2} \sqrt{2} y_{2,3} \dots y_{d,d} \right]^\top.$$

In general, the identity matrix is chosen for P [?, ?]. Thus, the vectorized covariance matrix is given by

$$\mathbf{c} = \mathrm{vec}_I(\log(C)).$$

The vector $\mathbf{c}$ is finally normalized with the signed square rooting normalization and ℓ_2 normalization.

In our implementation, we use the VGG-M network [?] for the pre-trained CNN. The final feature vector of each depth-buffer image is obtained by concatenating the fully connected layer activations and FMCD extracted from the first convolutional layer. To compare two 3D models, we apply the Hungarian method [?] to all pair Euclidean distances between their feature vectors. We output models by which the distance to the query is less than 0.9. We call this run DB-FMCD-FUL.

Moreover, we calculate ranking scores by using the Locally Constrained Diffusion Ranking (LCDR) [?], and call this run DB-FMCD-FUL-LCDR. In this run, we output models for which the ranking score is more than 0.0001.

6. Results

The summary results for all participating methods are given in Table 1. The corresponding precision-recall plots are given in Figure 6 for the normal dataset splits, and in Figure 7 for the perturbed

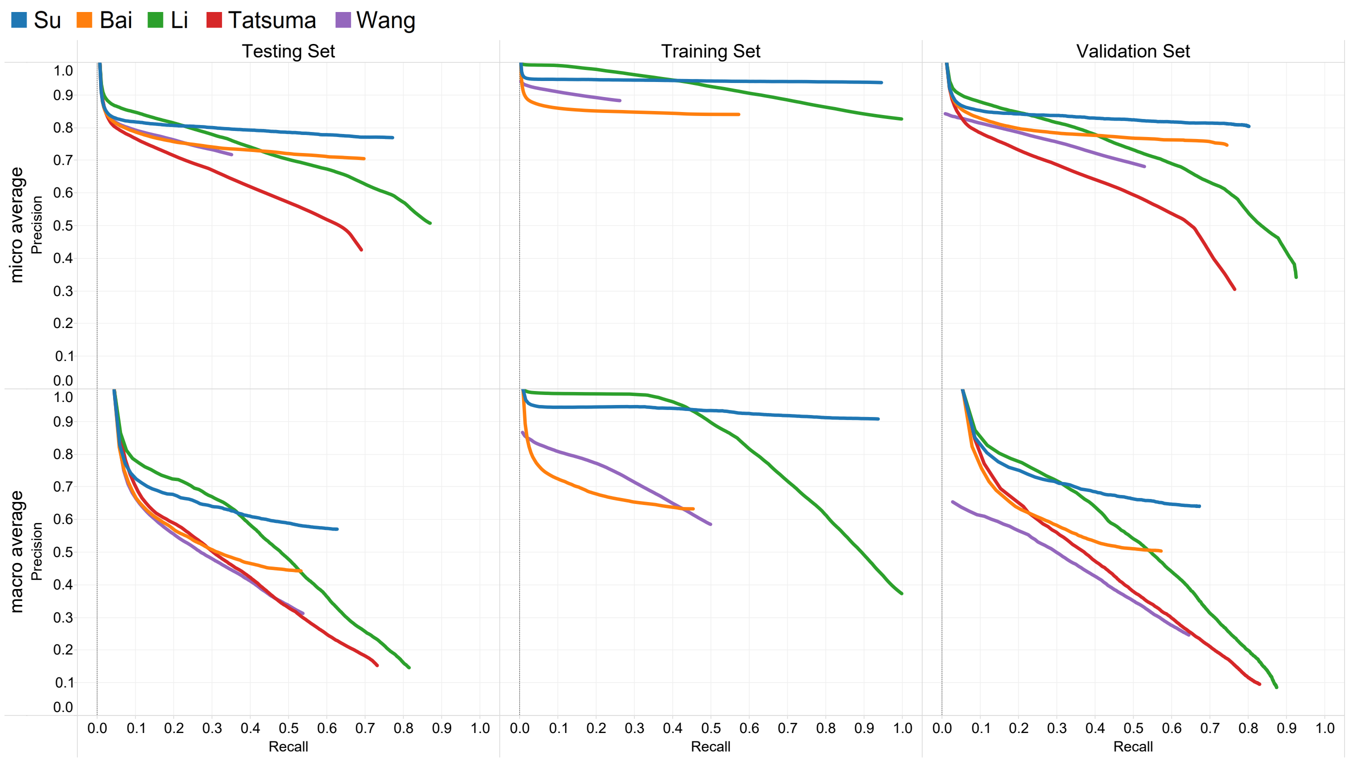

Figure 6: *Precision-recall plots for all participating teams and methods, on the normal dataset splits.*

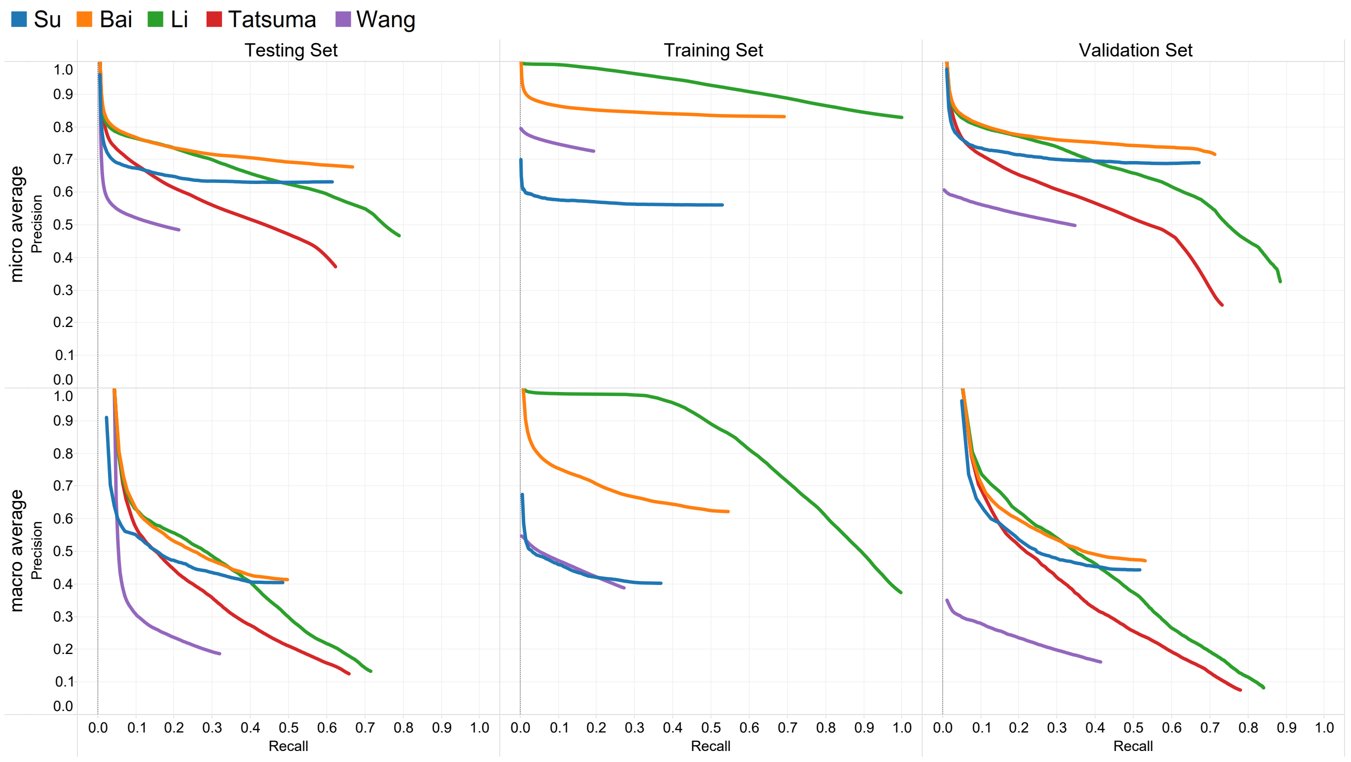

Figure 7: *Precision-recall plots for all participating teams and methods, on the perturbed dataset splits.*

			micro					macro				
Dataset	Rotation	Method	P@N	R@N	F1@N	mAP	NDCG@N	P@N	R@N	F1@N	mAP	NDCG@N
Testing	Normal	Su	**0.770**	0.770	**0.764**	**0.873**	0.899	**0.571**	0.625	**0.575**	**0.817**	**0.880**
		Bai	0.706	0.695	0.689	0.825	0.896	0.444	0.531	0.454	0.740	0.850
		Li	0.508	**0.868**	0.582	0.829	**0.904**	0.147	**0.813**	0.201	0.711	0.846
		Wang	0.718	0.350	0.391	0.823	0.886	0.313	0.536	0.286	0.661	0.820
		Tatsuma	0.427	0.689	0.472	0.728	0.875	0.154	0.730	0.203	0.596	0.806
Validation	Normal	Su	**0.805**	0.800	**0.798**	**0.910**	**0.938**	**0.641**	0.671	**0.642**	**0.879**	**0.920**
		Bai	0.747	0.743	0.736	0.872	0.929	0.504	0.571	0.516	0.817	0.889
		Li	0.343	**0.924**	0.443	0.861	0.930	0.087	**0.873**	0.132	0.742	0.854
		Wang	0.682	0.527	0.488	0.812	0.881	0.247	0.643	0.266	0.575	0.712
		Tatsuma	0.306	0.763	0.378	0.722	0.886	0.096	0.828	0.140	0.601	0.801
Training	Normal	Su	**0.939**	0.944	**0.941**	0.964	0.923	**0.909**	0.935	**0.921**	0.964	0.947
		Bai	0.841	0.571	0.620	0.907	0.912	0.634	0.452	0.472	0.815	0.891
		Li	0.827	**0.996**	0.864	**0.990**	**0.978**	0.374	**0.997**	0.460	**0.982**	**0.986**
		Wang	0.884	0.260	0.363	0.917	0.891	0.586	0.497	0.428	0.775	0.863
Testing	Perturbed	Su	0.632	0.613	0.612	0.734	0.843	0.405	0.484	0.416	0.662	0.793
		Bai	**0.678**	0.667	**0.661**	**0.811**	**0.889**	**0.414**	0.496	**0.423**	**0.730**	**0.843**
		Li	0.467	**0.789**	0.534	0.749	0.865	0.134	**0.714**	0.182	0.579	0.767
		Wang	0.486	0.212	0.246	0.600	0.776	0.187	0.319	0.163	0.478	0.695
		Tatsuma	0.372	0.622	0.413	0.638	0.838	0.126	0.657	0.166	0.493	0.743
Validation	Perturbed	Su	0.691	0.671	0.673	0.813	0.894	0.444	0.516	0.456	0.764	0.853
		Bai	**0.717**	0.712	**0.706**	**0.857**	**0.921**	**0.472**	0.530	**0.481**	**0.803**	**0.878**
		Li	0.327	**0.884**	0.423	0.776	0.891	0.083	**0.840**	0.126	0.600	0.775
		Wang	0.499	0.346	0.336	0.571	0.736	0.162	0.413	0.167	0.292	0.498
		Tatsuma	0.255	0.731	0.332	0.651	0.853	0.076	0.779	0.114	0.509	0.743
Training	Perturbed	Su	0.562	0.529	0.540	0.608	0.737	0.403	0.368	0.376	0.529	0.672
		Bai	**0.832**	0.691	0.717	0.898	0.910	**0.623**	0.544	**0.526**	0.804	0.888
		Li	0.830	**0.998**	**0.867**	**0.991**	**0.980**	0.375	**0.996**	0.461	**0.979**	**0.984**
		Wang	0.726	0.192	0.282	0.763	0.801	0.389	0.271	0.254	0.486	0.658

Table 1: *Summary table of evaluation metrics for all participating teams and methods, on all competition dataset splits.*

		micro + macro average				
Rotation	Method	P@N	R@N	F1@N	mAP	NDCG@N
Normal	Su	**0.671**	0.698	**0.669**	**0.845**	**0.890**
	Bai	0.575	0.613	0.572	0.783	0.873
	Li	0.328	**0.841**	0.392	0.770	0.875
	Wang	0.516	0.443	0.338	0.742	0.853
	Tatsuma	0.290	0.709	0.338	0.662	0.841
Perturbed	Bai	**0.546**	0.581	**0.542**	**0.770**	**0.866**
	Su	0.519	0.549	0.514	0.698	0.818
	Li	0.301	**0.751**	0.358	0.664	0.816
	Wang	0.337	0.265	0.205	0.539	0.736
	Tatsuma	0.249	0.640	0.290	0.566	0.791

Table 2: *Participating methods ranked on normal and perturbed test datasets. The rank is computed by the average of the micro and macro mAP.*

dataset splits. We can get several observations from the summary evaluation metrics.

Firstly, all participating teams adopt projection-based approaches, i.e., they render 3D meshes into RGB images or depth images and then make the comparison over renderings. In addition, deep learning methods are used by all teams to extract strong image features. Clearly, shape retrieval is benefiting from the development of computer vision techniques, in particular, the recent progress of deep learning methods.

Secondly, the overall retrieval performance on the normal test dataset as measured by micro-averaged mAP and NDCG values is fairly high (in the 70-80% range for mAP and in the high 80% to 90% range for NDCG). The mAP measure is highly correlated with the AUC giving a sense of the area under the precision-recall curve for each method's chosen retrieval list length. Similarly, NDCG evaluates the overall relevance ranking of the retrieved model list against an ideal ranking of the same retrieved list. All methods perform well when judged on the chosen retrieval list lengths. However, the methods have very different trade offs of performance when we consider precision and recall. For example, **Li** have the highest R@N value but a fairly low P@N value, whereas **Wang** have a high P@N value and significantly lower R@N value. Other methods are more balanced in precision and recall.

When using macro-averaging instead, we no longer adjust for class sizes, thus giving proportionally more weight to synsets with fewer models. The same relative patterns between methods as observed in micro-averaging hold with macro-averaging as well, but the overall performance for all methods is significantly lower, as expected (F1@N scores drop from a high in the 70% range to a high in the 50% range).

The performance on the normal validation set is similar and generally a bit higher than the test set, as expected. The performance of the methods on the training set indicates that most likely **Su** and **Li** are starting to be data-starved and perhaps over-fitting (most evaluation metrics are in the mid 90% range). Other methods have lower metrics on training indicating that they could perhaps still extract more utility out of the data.

When contrasting the normal dataset with the perturbed dataset we see that, as expected, there is a significant drop in performance for all methods, except for **Bai** which in general experience a much smaller drop. Overall, **Bai** seem to have the best performance on the perturbed dataset, very likely due to the views uniformly sampled on a sphere.

Interestingly, for the perturbed datasets, validation performance is lower than test performance, and only **Li** seem to be fully data-starved and overfitting in the training set performance. Other methods still seem to have significant room to improve even in the training set for the perturbed data. This suggests that better ways of handling unknown orientations in the 3D models might be beneficial.

We also see two different ways to handle subcategories. **Su** trains separate models for categories and subcategories, while **Li** tries to model them jointly. In our benchmark, only NDCG considers subcategories. From Table 1, in micro evaluation, **Li** has better scores than **Su** in NDCG, despite the aforementioned overfitting problem. It indicates that the joint training may be better in modeling subcategories.

Overall, deep learning models achieve good performance in this competition. Projection-based inputs are used in all of the submissions, because it is easy to make use of additional image data to help learn features for 3D shapes. Because of this, the models are based on image classification models and they are not built from scratch for the 3D problems. In the case of unknown rotations, it is helpful to sample views on a sphere instead of circle (as indicated by the advantage of the Bai team's approach over others in the perturbed data. If subcategories are considered, it is potentially better to model category and subcategories jointly. As the projection-based models are popular in this competition, we hope to see exploration into a broader variety of models in the future.

7. Conclusion

In this paper, we compared several methods for 3D model retrieval on a new large-scale dataset of 3D models. All methods use some form of neural network architecture and the performance on a consistently aligned version of the dataset is fairly high. In contrast, there is space for improvement when handling models with unknown orientations.

The findings of the competition demonstrate that view-based methods leveraging recent advances in deep learning can provide very strong performance in 3D shape retrieval tasks. Furthermore, fine-tuning neural networks that were originally trained on external image datasets seems to be another common theme in participant methods. However, the sensitivity of view-based methods to consistent model orientations was evident in the drastically lower performance on a dataset with random model orientations. Future work might investigate methods for improving performance when the consistent orientation assumption cannot be made. Hybrid approaches integrating view-based and geometry-based approaches are another interesting avenue for future research.