SE3-Unet - Visione Artificiale

Mattia Ferraretto [00072A]

Academic Year 2023/2024

Contents

1 Problem statement **3**

2 Introduction **3**

3 SE3-Transformer **3**

4 Dataset **4**

5 3D - Pooling **5**

6 3D - Up sampling **5**

7 Model architecture **6**

8 Experiments and results **7**

8.1 Hyper-parameters settings 8

8.2 Results 9

9 Conclusion **13**

1 Problem statement

The project consists in developing a novel neural network architecture with the aim to address heatmap regression problem. The idea is to replace the classic convolution block in U-net architecture [1] with a SE3-Transformer block [2], in such a way to make it resilient to 3D roto-translation. Then, the network is trained on FaceScape [3], a large-scale detailed 3D face dataset, with the objective of inference face landmarks.

2 Introduction

Before diving into SE3-Transformer let me introduce two fundamental concepts: Equivariance and Group Representation.

Equivariance Give a set of transformation $T_g : \mathcal{V} \to \mathcal{V}$ for $g \in G$ where G is an abstract group, a function $\phi : \mathcal{V} \to \mathcal{Y}$ is called equivariant if for every g there exist a transformations $S_g : \mathcal{Y} \to \mathcal{Y}$ such that

$$S_g\left[\phi(v)\right] = \phi(T_g\left[v\right]) \qquad \text{for all } \; g \in G,\, v \in \mathcal{V} \tag{1}$$

Where g can be seen as parameters describing the transformation. In the equivariance literature, deep networks are built from interleaved linear maps ϕ and equivariant nonlinearities. In the case of 3D roto-translations it has already been shown that a suitable structure for ϕ is a *tensor filed network*.

Group Representations In general, the transformation (T_g, S_g) are called *group representations*. Formally, a group representation $\rho : G \to GL(N)$ is a map from a group G to the set of $N \times N$ invertible matrices $GL(N)$. In the case of 3D rotations $G = SO(3)$ we have two interesting properties: 1) its representations are orthogonal matrices, 2) all representations can be decomposed as

$$\rho(g) = Q^T \left[\bigoplus_{\ell} \mathbf{D}_\ell(g)\right] Q \tag{2}$$

where Q is an orthogonal, $N \times N$, change-of-basis matrix; each $\mathbf{D}_\ell$ for $\ell = 0, 1, 2, \ldots$ is a $(2\ell + 1) \times (2\ell + 1)$ matrix known as a Wigner-D matrix; and the $\bigoplus$ is the *direct sum* or concatenation of matrices along the diagonal. Wigner-D matrices are *irreducible representations* of SO(3). Vectors transforming according to $\mathbf{D}_\ell$, are called type-ℓ vectors. Type-0 are invariant under rotations and type-1 vectors rotate according to 3D rotation matrices.[1]

3 SE3-Transformer

In the SE3-Transformer, the concept of **neighbourhoods** is introduced. Given a point cloud $\{(\mathbf{x_i}, \mathbf{f_i})\}$, the idea is to define a set of neighborhoods $\mathcal{N}_i \subseteq \{1, \ldots, N\}$ centered on each point i. These neighbourhoods are computed either via the nearest-neighbours methods or may already be defined. The introduction of this neighbourhoods is justified by the reduced cost of computing the attention mechanism since it's quadratic with respect to the input.

[1]For more details look at the original paper. [2]

Architecture The SE(3)-Transformer itself consist of three components: 1) edge-wise attention weights α_{ij}, constructed to be SE(3)-invariant on each edge ij; 2) edge-wise SE(3)-equivariant value messages, propagating information between nodes (as in Tensor Field Network); 3) a linear/attentive self-interaction layer. Attention is performed on a per-neighbourhood basis as follow:

$$\mathbf{f}^{\ell}_{\text{out},i} = \underbrace{\mathbf{W}^{\ell\ell}_{V}\mathbf{f}^{\ell}_{\text{in},i}}_{③\ \text{self-interaction}} + \sum_{k\geq 0}\sum_{j\in\mathcal{N}_i\setminus i} \underbrace{\alpha_{ij}}_{①\ \text{attention}} \underbrace{\mathbf{W}^{\ell k}_{V}\left(\mathbf{x}_j - \mathbf{x}_i\right)\mathbf{f}^{k}_{\text{in},j}}_{②\ \text{value message}} \tag{3}$$

In this equation, if we remove the attention weights then we have a tensor field convolution, and if we instead remove the dependence of $\mathbf{W}_V$ on $(\mathbf{x}_j - \mathbf{x}_i)$, we have a conventional attention mechanism. Provided that the attention weights α_{ij} are invariant, Eq. (3) is equivariant to SE(3)-transformations. Invariant attention weights can be computed with a dot-product attention structure shown in Eq. (4). This mechanism consist of a normalised inner product between a query vector $\mathbf{q}_i$ at node i and a set of key vectors $\{\mathbf{k}_{ij}\}_{j\in\mathcal{N}_i}$ along each edge ij in the neighbourhood $\mathcal{N}_i$ where

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^{\top}\mathbf{k}_{ij})}{\sum_{j'\in\mathcal{N}_i\setminus i}\exp(\mathbf{q}_i^{\top}\mathbf{k}_{ij'})}, \quad \mathbf{q}_i = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}^{\ell k}_{Q}\mathbf{f}^{k}_{\text{in},i}, \quad \mathbf{k}_{ij} = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}^{\ell k}_{K}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}^{k}_{\text{in},j} \tag{4}$$

$\bigoplus$ is the direct sum, i.e. vector concatenation in this instance. The linear embedding matrices $\mathbf{W}^{\ell k}_{Q}$ and $\mathbf{W}^{\ell k}_{K}(\mathbf{x}_j - \mathbf{x}_i)$ are TFN type. The attention weights α_{ij} are invariant for the following reason: when input features $\{\mathbf{f}_{in,j}\}$ are SO(3)-equivariant, then the query $\mathbf{q}_i$ and key vectors $\{\mathbf{k}_{ij}\}$ are also SE(3)-equivariant, since the linear embeddings matrices are of TFN type.[2]

4 Dataset

In this section, I'm going to introduce the dataset used for this work, the pre-processing phase, and finally, the resolution reduction technique applied to the data.

Facescape The dataset used for this work is called Fascescape. It consists of a collection of 18,760 high resolution 3D faces with 20 different expressions. Each face has been acquired by using a multi-view 3D reconstruction system composed by 68 DSLR cameras, 30 of which capture 8K images focusing on front side, and the other cameras capture 4K level images for the side parts. Subjects scanned are people between 16 and 70 years old and mostly from Asia. Additional information for each subject are recorded such as gender and job (by voluntary). Along with the dataset a set of landmark points are provided, each landmark identify a specific portion of a face. I focused my work only on a subset of faces (669 for the training set and 167 for the test set) and a single expression: Neutral.

Point cloud resolution reduction Since FaceScape data are point clouds having a resolution 8192 points, due to the limited compute available they have been reduced to a manageable resolution (test were performed with 1024 points). To reduce the resolution the idea is to apply Farthest Point Sampling (FPS) algorithm which aims to select a representative subset of point from the original

[2]For additional detail read the original paper [2]

point cloud. Along with the point cloud reduction process, even each heatmap associated has been recomputed. In particular, with respect to each landmark i:

$$h_i = \exp\left(-\frac{d(x_i,\, x_j)^2}{2\sigma^2}\right) \tag{5}$$

where d is the Euclidean distance squared, x_i is the 3D coordinates of i-th landmark, x_j correspond to a point in the point cloud, and in the end, σ is a parameter shaping the Gaussian's bell. Repeating this transformation for each point, the resulting h_i is a vector of shape $1 \times N$, where N is the number of points, corresponding to the heatmap for the landmark. In my setting, σ is set to 0.095.

Pre-processing Before testing model performance, the dataset has been pre-processed by two different approaches: 1) by applying random 3D roto-translations and then re-aligning data using ICP; 2) by simply applying random 3D roto-tranaslation. The main objective of this kind of pre-process is to simulate a scenario in the wild where data are not acquired in a controlled environment. The second point is about to evaluate the capacity of the SE(3)-Transformer to be resilient to rotation and translation in the field of 3D data compared to other approaches.

5 3D - Pooling

In the classic CNN framework the pooling operation consists in replacing the value at certain location with a summary statistic of nearby values. In the literature the are different types of pooling functions, such as Max Pooling, Average Pooling, and they work well in grid-like data (e.g. 2D images) where a neighbourhood is clearly defined. However, 3D data has not a fixed structure and a pooling operation must take into account such fact for working correctly in this scenarios. For simulate traditional pooling my idea is to select a subset of points using FPS according to a specific pooling ratio, aggregate features based on their neighbourhoods (neighbourhood can be defined by k-nn methods or already pre-computed), and finally discard points not selected by FPS. The resulting point cloud will be one with a resolution reduced by the pooling ratio and features aggregated in the maintained points. Of course, also in that case, different kind of pooling can be applied (Max, Average, Sum ecc..). In the code base provide Sum, Max and Average are supported, as well as, multi-channel pooling.

6 3D - Up sampling

Inverse pooling operation in CNN is called transpose convolution where a pooled input is reported to the original resolution with the help of a learnable kernel. To replicate this operation in a 3D setting my idea is to estimate features by applying a technique called Inverse Distance Weighting (IDW). IDW assumes that the influence of a known feature decreases as the distance to the unsampled location increases. Therefore, the weights assigned to each sampled feature are inversely proportional to their distance from the unsampled location. Formally speaking:

1. The first step consist in computing the weights:

   $$w_j = \frac{1}{d(x_i,\, x_j)^p} \tag{6}$$

   where d is the Euclidean distance between x_i and x_j, x_i is the location in which we are interested in estimating the features, x_j is a location of a data point belonging to the neighbourhood of

x_i, and p is the power parameter controlling how rapidly the influence of a point decreases as the distance increases.

2. Once the weights are computed, the features for x_i are estimated as the weighted average of its neighbourhood:

$$f_i = \frac{\sum_{j \in \mathcal{N}_i} w_j f_j}{\sum_{j \in \mathcal{N}_i} w_j} \tag{7}$$

where w_j is the weight associated to f_j and denominator is a normalization term.

To make this approach suitable to the problem setting, on top of point clouds a knn-graph has been built based on the distances of points, then during the forward pass of the network the structure of the graph has been kept at each level (for more datails see the next section).

7 Model architecture

As mentioned in Section (1) the architecture defined is similar to a U-net where instead to have classic convolutional layers SE(3)-Transformer layers are used in place.

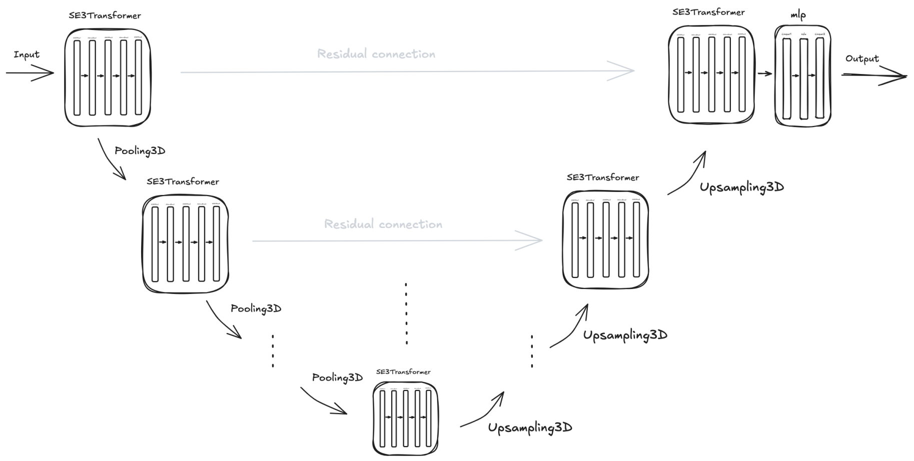


Figure 1: SE(3)-Unet architecture followed by a classification head.

U-net U-net architecture consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3 convolution (unpadded convolutions), each followed by a rectified linear unit (ReLU) and 2×2 max pooling operation with stride 2 for downsampling. At each downsampling step authors double the numbers of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2×2 convolution that halves the number of features channels, a concatenation with the correspondingly cropped features map from the contracting path, and two 3×3 convolution, each followed by a ReLU. At the final layer a 1×1 convolution is applied.[1]

SE(3)-Unet Similarly to the original U-net, SE(3)-Unet has a contracting path (left side) and an expansive path (right side). In this architecture, the contracting path consists of a repeated application of a SE3Transformer, where it composed by two GSE3Res (Graph attention block with SE(3)-equivariance and skip connection) followed by a GNormBias (Norm-based SE(3)-equivariant nonlinearity with only learned biases), and another GSE3Res. Finally, Pooling3D operation with a specified polling ratio is applied. As a strong difference with respect to the original network is that the number of channel is fixed to 5. Every step in the expansive path consists in a Upsampling3D operation followed by the application of a SE3Transformer. At each up sampling step, after the SE3Transformer layer transformation, residual connections coming from the contracting path are summed. Finally, features are mapped to the target classes using an MLP composed as a linear layer, followed by a ReLU activation, followed by another linear layer.

Equivariance tests Once architecture has been defined equivariance tests have been applied. The test consists in: (1) taking a point cloud; (2) applying a transformation such as a rotation/translation or both, to a copy of the sampled point cloud; (3) passing both points (transformed and original ones) through the model; (4) applying the same transformation to the output of the point cloud fed into the model without any transformation; (5) checking if the result obtained is the same. From this experiments, I found out that the model is resilient to rotation, bot not to translation, in fact, even roto-translation are affected from the issue because this operation consists in applying a rotation then a translation. I tried to investigate this problem but I didn't find the cause. You can find equivariance test notebook in the Github repo.

8 Experiments and results

In this section I'm going to describe the two experiments performed and results obtained, but first let me introduce the environment. Experiments have been made in Amazon Sagemaker Studio Lab free tier. It offers a CPU 4-core with 12 GB of RAM and a single Nvidia T4 with 16 GB of VRAM. A Python virtual environment has been defined using Conda where libraries required to run the code have been installed. In details, the most important libraries and their respective versions are: python v3.11.9, dgl v2.4.0+cu118, torch v2.1.0, torch-geometric v2.5.3, open3d v0.18.0.

Since the challenging optimization problem, traditional loss functions, such as Mean Squared Error (MSE), struggle with the imbalance between foreground (high-intensity) and background (low-intensity) pixels, leading to blurry heatmaps and inaccurate predictions. To address these limitations, **Adaptive Wing Loss (AWing Loss)** was introduced as an improvement over existing loss functions by dynamically adjusting its curvature based on pixel intensity.

The AWing Loss is designed to **increase sensitivity to small errors in foreground pixels** while reducing the influence of background errors. This is crucial for accurate landmark localization, as even minor errors in high-intensity regions can significantly impact prediction accuracy. The formulation of the loss function is as follows:

$$\mathcal{L}_{AWing}(y, \hat{y}) = \begin{cases} \omega \ln \left(1 + \left|\frac{y-\hat{y}}{\epsilon}\right|^{\alpha - y}\right), & \text{if } |y-\hat{y}| < \theta \\ A|y-\hat{y}| - C, & \text{otherwise} \end{cases}$$

where:

- y and $\hat{y}$ are the ground truth and predicted heatmap values, respectively.

- ω, θ, ϵ, and α are hyperparameters controlling the shape and transition of the loss.

To ensure smoothness and continuity at $|y - \hat{y}| = \theta$, the parameters A and C are defined as:

$$A = \omega \left(\frac{1}{1 + (\theta/\epsilon)^{\alpha - y}} \right) (\alpha - y) \left(\frac{\theta}{\epsilon} \right)^{\alpha - y - 1} \frac{1}{\epsilon}$$

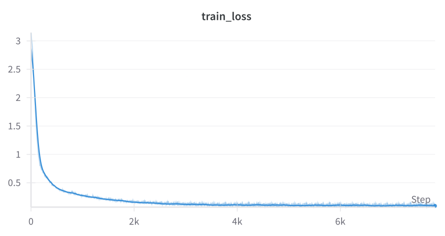
$$C = \theta A - \omega \ln \left(1 + \left(\frac{\theta}{\epsilon} \right)^{\alpha - y} \right)$$

In the experiment performed I used the best hyperparameters settings according to its paper [4].

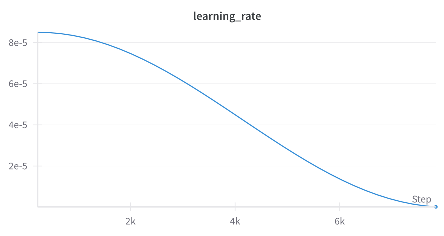
SE(3)-Unet configuration details SE(3)-Unet has been defined with a single layer, 5 hidden channels, a self-interaction in hidden layers of type 1×1 (Graph Linear SE(3)-equivariant layer, equivalent to a 1x1 convolution), a self-attentive interaction in the final layer, a Sum pooling operation with a ratio of 0.35, and finally, an output map of 68 features (I used a single layer due to the limited compute available in the Amazon free tier, while I chose 5 hidden channels according to [2] in the point cloud segmentation experiment).

8.1 Hyper-parameters settings

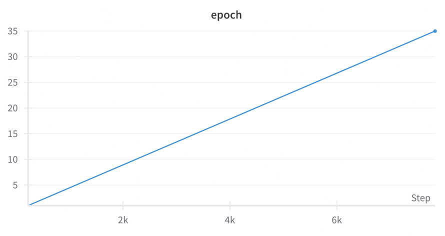
The network has been trained for 35 epochs (roughly 8K steps) with a batch size of 3 and starting learning rate of $8.5e{-5}$. Learning rate has been decreased according to `CosineAnnealingWarmRestarts` scheduler until reaching a value close to 0 at the end of the training. As already mentioned, the hyper-parameters for the `AWingLoss` are set to the optimal one proposed in the paper: $\omega = 14$, $\theta = 0.5$, $\epsilon = 1$, $\alpha = 2.1$. Here follow main training curves:



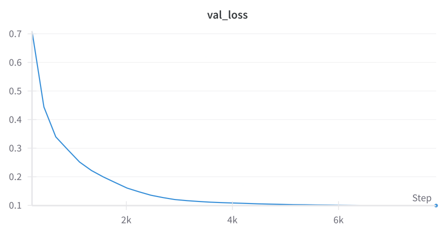
(a) Training loss



(b) Learning rate



(c) Epochs



(d) Validation loss

Figure 2: Training metrics

With this hyper-parameters setting I obtained good training curves during the training as shown in Figure 2. In fact training loss is pretty stable and converges to zero as epochs grows, while the validation loss follows the same pattern: as training goes on as model becomes better over the validation set to inference heatmaps.

8.2 Results

To test the model performance, in both dataset settings, I decided to use Intersection over the Union metric, to evaluate how much heatmaps are aligned with respect to groundtruth. Results are reported below, for each dataset pre-processing. In the Github repo are reported IoU charts at different thresholds for each landmark.

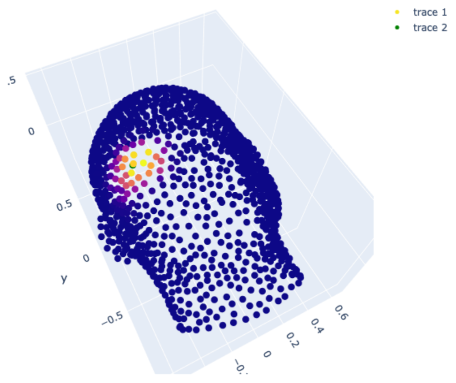
Table 1: IoU results with a threshold of 0.2

#	**None**	$\textbf{ICP}_{\textbf{r}}$	$\textbf{ICP}_{\textbf{t}}$	$\textbf{ICP}_{\textbf{rt}}$	$\textbf{ST}_{\textbf{r}}$	$\textbf{ST}_{\textbf{t}}$	$\textbf{ST}_{\textbf{rt}}$
0	0.5444	0.5395	0.5446	0.5352	0.0144	0.0032	0.001
1	0.6316	0.5872	0.5823	0.5806	0.0189	0.0042	0.0006
2	0.6509	0.5973	0.5965	0.5985	0.0142	0.0058	**0**
3	0.6483	0.5801	0.5714	0.5765	0.0104	0.0075	0
4	0.5774	0.5019	0.486	0.4991	0.0073	0.0087	0
5	0.5022	0.4226	0.4072	0.4182	0.0053	0.0073	0
6	0.4149	**0.3495**	**0.3351**	**0.3497**	0.0094	0.004	0
7	0.4182	0.3833	0.3733	0.3803	0.0087	0.0035	0
8	0.4027	0.3763	0.3809	0.376	0.0086	0.0054	0.0011
9	0.3765	0.3894	0.3956	0.3917	**0.004**	0.0054	0.0016
10	0.3858	0.4243	0.4399	0.4278	0.0048	0.0036	0.0003
11	0.3834	0.4357	0.4581	0.4475	0.0082	0.0023	0.0026
12	0.3985	0.4468	0.4695	0.451	0.015	0.0033	0.0004
13	0.4352	0.4657	0.4786	0.4684	0.0207	0.0058	0.0003
14	0.5193	0.5475	0.5539	0.5469	0.0167	0.0099	0.0006
15	0.5845	0.602	0.6079	0.6083	0.015	0.0136	0.0012
16	0.6628	0.6617	0.6491	0.6663	0.0164	0.0154	0
17	0.5161	0.519	0.5181	0.5153	0.0282	0.0026	0.0003
18	0.4681	0.4714	0.4719	0.4709	0.0304	**0.0013**	0
19	0.48	0.4861	0.4877	0.489	0.0232	0.0024	0.0012
20	0.5038	0.5217	0.5138	0.5153	0.0249	0.0025	0.0017
21	0.5933	0.605	0.5969	0.6002	0.0332	0.0051	0.0005
22	0.5928	0.6267	0.6	0.6251	0.0269	0.0118	0.0204
23	0.5298	0.5597	0.5299	0.5522	0.0317	0.0195	0.023
24	0.4728	0.4988	0.4718	0.4964	0.0312	0.0389	0.0232
25	0.4706	0.497	0.4667	0.4905	0.0247	0.0469	0.0224
26	0.4945	0.5232	0.4926	0.5255	0.0156	**0.0502**	0.0198

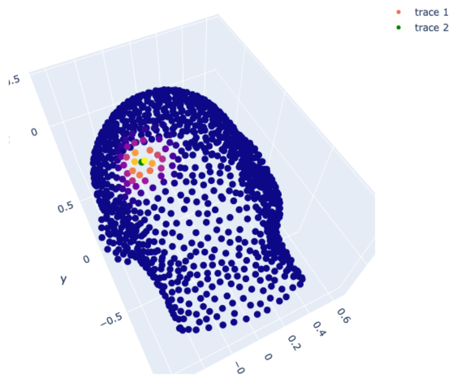
#	**None**	$\mathbf{ICP_r}$	$\mathbf{ICP_t}$	$\mathbf{ICP_{rt}}$	$\mathbf{ST_r}$	$\mathbf{ST_t}$	$\mathbf{ST_{rt}}$
27	0.6409	0.6392	0.6362	0.6467	0.0365	0.0131	0.0007
28	0.6155	0.5995	0.6047	0.606	0.0409	0.0109	0.0001
29	0.6362	0.6007	0.606	0.6023	0.0318	0.0151	0
30	0.5505	0.529	0.5358	0.5326	0.0263	0.0141	0
31	0.5583	0.527	0.5323	0.527	0.0209	0.0123	0.0004
32	0.5615	0.5288	0.5259	0.5255	0.0226	0.0144	0.0007
33	0.5928	0.567	0.5642	0.5687	0.027	0.0153	0.0008
34	0.5765	0.5512	0.5405	0.5468	0.0293	0.0193	0.0008
35	0.5952	0.5573	0.5582	0.5637	0.0324	0.0185	0.0002
36	0.6381	0.6232	0.6308	0.623	0.0465	0.0054	0.0001
37	0.6684	0.6495	0.6551	0.6556	0.0451	0.0049	0
38	0.6426	0.6369	0.6359	0.6407	0.0451	0.0056	0
39	0.6135	0.6108	0.6034	0.6078	0.0466	0.0077	0
40	0.6717	0.6495	0.6527	0.6525	0.0447	0.0063	0
41	$\overline{\mathbf{0.6881}}$	$\overline{\mathbf{0.6743}}$	$\overline{\mathbf{0.6799}}$	0.6732	0.0478	0.0058	0.0001
42	0.6421	0.6407	0.6213	0.6366	0.0417	0.0175	0.0124
43	0.6577	0.6647	0.6421	0.6631	0.0504	0.0269	0.021
44	0.6372	0.6518	0.6212	0.6538	$\overline{\mathbf{0.0521}}$	0.0379	$\overline{\mathbf{0.0242}}$
45	0.606	0.6169	0.5829	0.6158	0.0315	0.0369	0.0188
46	0.661	0.6678	0.6333	0.6656	0.043	0.0341	0.0187
47	0.6687	0.6721	0.6573	$\overline{\mathbf{0.682}}$	0.0495	0.0259	0.016
48	0.433	0.4027	0.4049	0.4026	0.0174	0.0101	0.0002
49	0.4746	0.426	0.4272	0.431	0.0125	0.0131	0.0005
50	0.4573	0.4119	0.4085	0.4137	0.0113	0.013	0.0018
51	0.4311	0.3978	0.3961	0.3979	0.014	0.0151	0.0036
52	0.4407	0.4041	0.4054	0.4105	0.0146	0.0162	0.003
53	0.4445	0.4181	0.4148	0.417	0.0121	0.0148	0.0014
54	$\underline{\mathbf{0.3692}}$	0.3705	0.376	0.3695	0.015	0.0127	0
55	0.3749	0.367	0.3753	0.3705	0.0106	0.0136	0.0007
56	0.3726	0.356	0.3641	0.3638	0.0085	0.0136	0.0009
57	0.4326	0.4074	0.4124	0.4097	0.0088	0.0126	0.001
58	0.4584	0.4243	0.432	0.4299	0.0095	0.011	0.0003
59	0.4745	0.4396	0.439	0.4473	0.0135	0.0108	0
60	0.4768	0.4414	0.4447	0.4473	0.0186	0.0107	0.0002
61	0.4605	0.4212	0.4166	0.4205	0.01	0.013	0.0018
62	0.4325	0.3934	0.3891	0.3902	0.0101	0.0164	0.0028
63	0.4069	0.3757	0.3732	0.3777	0.0098	0.0168	0.0034
64	0.3798	0.3731	0.385	0.3818	0.0124	0.0135	0.0001

#	None	$\mathbf{ICP_r}$	$\mathbf{ICP_t}$	$\mathbf{ICP_{rt}}$	$\mathbf{ST_r}$	$\mathbf{ST_t}$	$\mathbf{ST_{rt}}$
65	0.4111	0.3797	0.38	0.3816	0.0093	0.0146	0.0016
66	0.4022	0.3722	0.3706	0.3748	0.009	0.0129	0.0013
67	0.434	0.4006	0.4015	0.4018	0.0093	0.0126	0.001

In Table 1 the best IoU score for each dataset is overlined while the worst is underlined. Of course, the higher the score, the better the alignment of the heatmap with groundtruth. Fianlly, to be clear, r stands for rotation, t stands for translation.



(a) No preprocessing best IoU



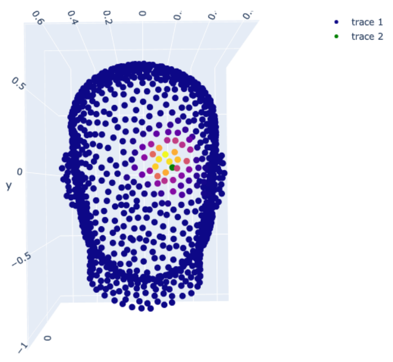
(b) No preprocessing groundtruth

Figure 3: Best heatmap obtained after training

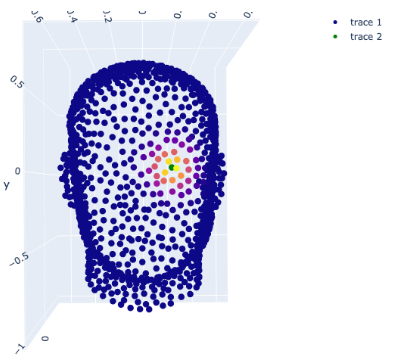
In Figure 3(a) is shown the heatmap obtained, for the best IoU score according to Table 1, over the test set with no pre-processing applied. In fact, as you can see, compared to the groundtruth (Figure 3(b)) the result is pretty good with the heatmap aligned to the groundtruth indicating the landmark of interest.

In Figure 4 are shown heatmaps obtained during test inference when different kinds of pre-processings have been applied. In particular, the first row, Figure 4(a) and Figure 4(b), shows the best landmark prediction compared to its groundtruth when the datasets was pre-processed by adding a random 3D roto-translation and then realigned by ICP algorithm. The plotted heatmap shows great model capability to predict a good landmark localization even when a little noise is introduced into the data. For simplicity, I omitted to display the single transformations (rotation and translation) because the roto-translation include both and according to IoU in Table 1 results are pretty similar.

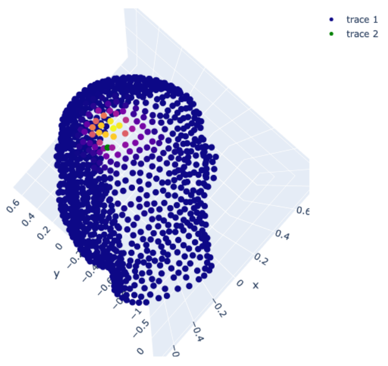
In the second row Figure 4(c) and Figure 4(d) is shown the heatmap predicted by the model and its corresponding groundtruth when a random 3D rotation is introduced into the test set without any realignment. As can be observed, the model has some difficulties to produce a centered heatmap where in fact it is a little bit higher than the expected one, confirming its IoU score of 0.0521.



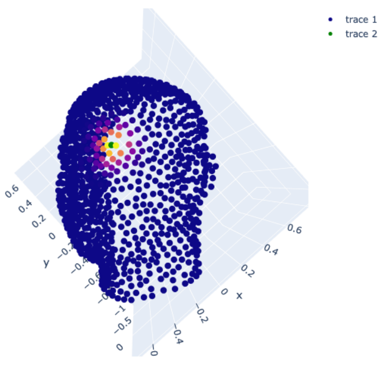
(a) $\mathrm{ICP_{rt}}$ best IoU



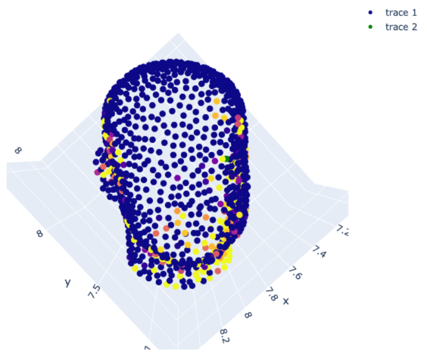
(b) $\mathrm{ICP_{rt}}$ groundtruth



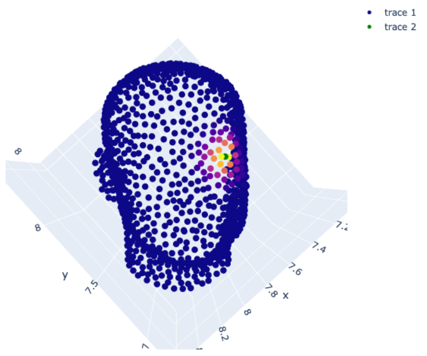
(c) $\mathrm{ST_{r}}$ best IoU



(d) $\mathrm{ST_{r}}$ groundtruth



(e) $\mathrm{ST_{rt}}$ best IoU



(f) $\mathrm{ST_{rt}}$ groundtruth

Figure 4: Heatmaps predicted after applying different test set pre-processing.

Finally, the last row, Figure 4(e) and Figure 4(f), displays the heatmaps and its corresponding groundtruth when a random 3D roto-translation is introduces into data and no realignment has been applied. According to its best IoU score, which is very low, the image confirms this poor result where no consistent heatmap is predicted by the model in such conditions.

9 Conclusion

In conclusion, I can state that the model is obtaining good results when no pre-processing is applied to the test data and also when little noise is injected into them through ICP pre-processing. In the moment some stronger noise is introduced by means of rotation and/or translations, on one side, it has some difficulties when a simple rotation is applied, I mean, a well defined heatmap is predicted but not very well aligned with the expected one. I think this behavior is due to the relative few steps performed during the training, in other words, I believe the model is not mature yet. On the other side, when two transformation, such as rotation and translation, are both applied together, the model completely fails to predict a consistent heatmap. In my opinion, this is related to the translation operation which breaks in somehow the equivariance, and this is also confirmed by the equivariance test performed before starting the training. For that reason I suggest to investigate further the cause of this strange behavior. I tried to figure out the issue by investigating how point clouds are processed by the data loader but every attempts failed to pass equivariance tests.

References

[1] U-Net: Convolutional Networks for Biomedical Image Segmentation, Olaf Ronneberger, Philipp Fischer, and Thomas Brox; https://arxiv.org/pdf/1505.04597

[2] SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks, Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, Max Welling; https://arxiv.org/pdf/2006.10503

[3] FaceScape: a Large-scale High Quality 3D Face Dataset and Detailed Riggable 3D Face Prediction, Haotian Yang, Hao Zhu, Yanru Wang, Mingkai Huang, Qiu Shen, Ruigang Yang, Xun Cao; https://arxiv.org/pdf/2003.13989

[4] Adaptive Wing Loss for Robust Face Alignment via Heatmap Regression, Xinyao Wang1, Liefeng Bo, Li Fuxin1; https://arxiv.org/pdf/1904.07399