Semester Project - 8 ECTS

3D Poses Recovery in Single-Particle Cryo-EM from Learned Pairwise Projection Distances

January 17, 2020

Author:
Jelena Banjac
Master of Data Science

Director:
Prof. Michael Unser

Advisors:
Laurène Donati (BIG)
Michaël Defferrard (LTS2)

Biomedical Imaging Group
Ecole Polytechnique Fédérale de Lausanne

Abstract

Single-particle cryo-electron microscopy (cryo-EM) is a technology that allows the observation and the high-resolution 3D structure determination of biomolecules. In this project, the goal is to estimate the angles at which we imaged the 2D projections from a given 3D protein. We developed deep learning models to estimate the angles from learned pairwise projection distances. We designed a two-step method: 1) distance estimation using a Siamese neural network to learn the distance between pairs of projections, and 2) angle recovery that includes a minimization scheme in order to estimate the angles at which each projection was taken. The current results obtained are discussed depending on different combination of approaches used and experimental conditions.

Keywords: deep learning, imaging, biology, cryo-EM, quaternion, siamese, distance metric, angle recovery

Contents

1 Introduction

Single-particle cryo-electron microscopy (cryo-EM) is a Nobel-prized technology that aims to reconstruct the 3D structure of various proteins at the atomic resolution[1]. The electron microscope (EM) first images at cryogenic temperature numerous replicates of the protein [2] positioned at various orientations (acquisition of 2D projections, see Fig.1(a)). Afterwards, using this set acquired projections as an input to advanced single-particle analysis algorithms, scientists are able to reconstruct a high-resolution 3D structure of the protein (reconstruction pipeline, see Fig.1(b),(c))[3].

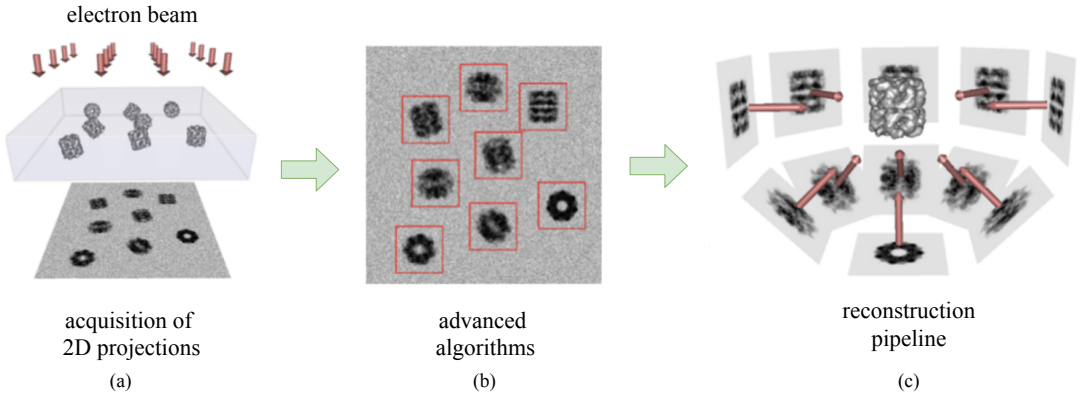

Figure 1: Cryo-EM reconstruction of a 3D protein structure from its 2D projections; (a) clones of the protein take random orientations in water and are then freezed; the electron beam is sent through the ice and captures the 2D image of the protein replicates; (b) imaging algorithms are used to segment the acquired 2D image to find separate 2D projections depending on the protein's orientation; (c) advanced 3D reconstruction algorithms are used to reconstruct the 3D structure of the protein. There, red arrows represent the angles at which the 2D projection images were taken. Image credits at [4]

The main challenge in single-particle cryo-EM reconstruction is that the angles at which the projection images were taken are unknown. The goal of this project is to recover the angles directly from the 2D projections. This is challenging as the 2D projection images are noisy, blurred, and can be shifted, see Fig.2. We present neural networks designed for estimating pairwise distances between pairs of 2D projections in order to recover these angles. To train our network, we simulate many projections of the protein.

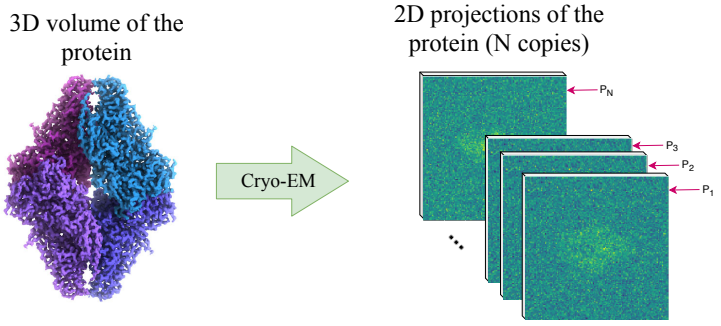

Figure 2: Cryo-EM allows observation of protein replicates positioned at N random orientations. From this, we get N 2D projections of this protein that are extremely noisy and blurred.

Through the following model, we consider that a cryo-EM measurement (i.e. 2D projection) $\mathbf{y}_i \in \mathbb{R}^M$ is acquired through

$$\mathbf{y}_i = \mathbf{C}_{\boldsymbol{\varphi}} \mathbf{S}_{\mathbf{t}} \mathbf{P}_{\theta_i} \mathbf{x} + \mathbf{n}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^V$ is the unknown 3D density map[5] (Coulomb potential). The operator $\mathbf{P}_{\theta_i} : \mathbb{R}^V \to \mathbb{R}^M$ is the projection along the 3D pose θ_i (i.e., the x-ray transform). The operator $\mathbf{S}_{\mathbf{t}} : \mathbb{R}^M \to \mathbb{R}^M$ is a shift of the projection by $\mathbf{t} = (t_1, t_2)$. The convolution operator $\mathbf{C}_{\boldsymbol{\varphi}} : \mathbb{R}^M \to \mathbb{R}^M$ models the microscope contrast transfer function (CTF) with parameters $\boldsymbol{\varphi} = (d_1, d_2, \alpha_{\text{ast}})$, that are, respectively, the defocus-major, the defocus-minor and the angle of astigmatism. Finally, $\mathbf{n} \in \mathbb{R}^M$ represents an additive noise. Our goal is then to recover the angles θ_i from every projection $\mathbf{y}_i$.

The report is structured as follows: In Chapter 2 we discuss the method developed in this project. In Chapter 3 we show the results obtained during the project. In Chapter 4 we discuss the results and propose future developments (section 4.1). We then conclude in Chapter 5.

2 Proposed Method

2.1 Rotation in SO(3) and Quaternions

There exist many ways to deal with rotations, such as Euler angles, rotation matrices, axis angles, quaternions [1]. In this work, we used the quaternion representation of rotations in 3D space. Quaternions can be seen as the extension of complex numbers[6]. Quaternions are generally represented in the form:

$$w + x\ \mathbf{i} + y\ \mathbf{j} + z\ \mathbf{k} \tag{2}$$

where w, x, y, and z are real numbers, and i, j, and k are the fundamental quaternion units.

Let's imagine that we have an electron beam positioned above a protein and we take a 2D projection image. Afterwards, we rotate the protein around $Z - Y - Z$ axes in 3D space with α, β, γ angles respectively and take another image. If the initial 2D projection results rotation angles (0, 0, 0), then the rotation that follows gives the projection with rotation angles (α, β, γ).

Now, let the quaternion $q_{Z,\alpha}$ be the rotation around Z axis with angle α, the quaternion $q_{Y,\beta}$ be the rotation around Y axis with angle β, and the quaternion $q_{Z,\gamma}$ be the rotation around Z axis with angle γ. The product of these three rotation quaternions[7] (a.k.a. Hamilton product[8]) are equivalent to the rotation $q_{Z,\alpha}$, followed by the rotation $q_{Y,\beta}$, and followed by the rotation $q_{Z,\gamma}$. Therefore:

$$q = q_{Z,\alpha} q_{Y,\beta} q_{Z,\gamma} \tag{3}$$

where q is a new quaternion presenting all three rotations combined in that order.

Quaternion Distance The distance $\theta \in [0, 2\pi]$ in radians between two quaternions q_1 and q_2[9] is given by:

$$\theta = 2 \arccos\left(|\langle q_1, q_2 \rangle|\right). \tag{4}$$

Quaternion-Euler angle conversions Since we are working with the quaternion representation of 3D rotations, it is necessary to handle the quaternion-Euler angle conversion in both directions. The equations for conversion can be seen in the Appendix chapter 6.

2.2 Pipeline Overview

We have a two-step method (see Fig.3):

1. **distance estimation (offline)** - once the distance metric between two projections is determined, it will be used for all future angle recoveries.

2. **angle recovery (online)** - will be ran with each new protein dataset of 2D projection images.

Assuming we have the distance estimation, we can only then do the angle recovery. These two types of optimizations are dual problems which we will further discus in the following text (section 2.3 and section 2.4).

[1] https://www.andre-gaschler.com/rotationconverter/

In addition to these two optimization, we will have a third optimization that will be used to estimate the error of the angle recovery. This optimization can be used only when the true rotation angles are provided and serves us just to see and understand more how far is estimated result from the optimal result (section 3.3).

The fact that we are dealing with the two-step method (distance estimation and angle recovery) can be seen in Fig.3. We start with 2D projection images and end up with their corresponding angles.

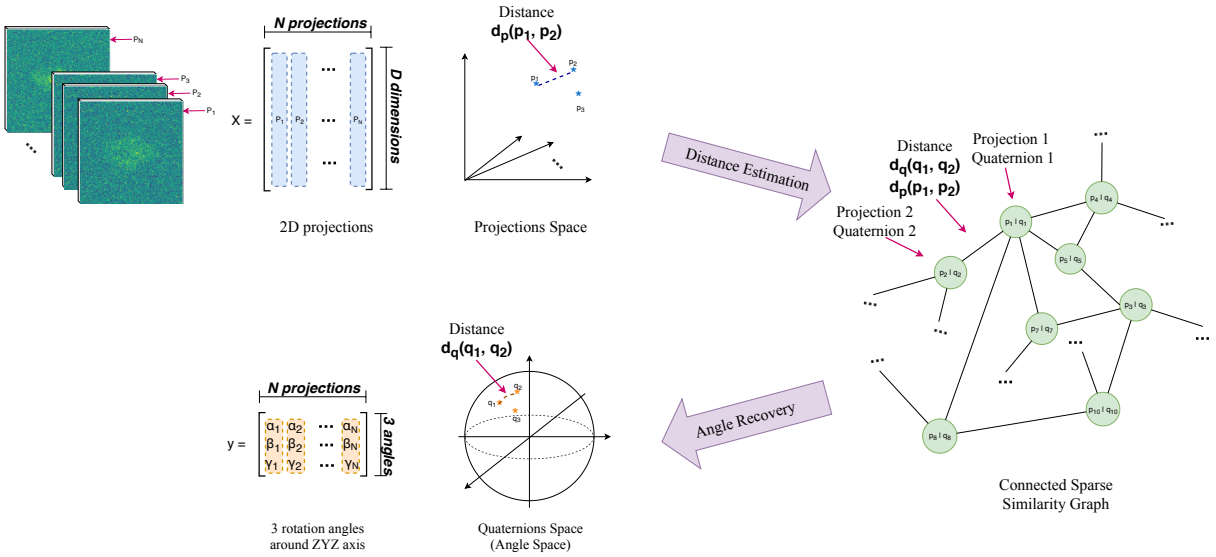

Figure 3: Project pipeline overview. With ***distance estimation*** method (i.e. estimating the function d_p) we create connected sparse similarity graph (see first purple arrow). The nodes represent the projection image vectors and its corresponding quaternion. The edges connect the closest nodes depending on their distance metric (projection distance and quaternion distance). The weights of the edges are the distances between two corresponding projection images p_1 and p_2 denoted as $d_p(p_1, p_2)$ and their corresponding distance between corresponding quaternions q_1 and q_2 denoted as $d_q(q_1, q_2)$. These quaternion values are unknown and our goal is to find them. We aim to reconstruct the angles from the learned pairwise distances using ***angle recovery*** method (see second purple arrow).

2.3 Distance Estimation (Offline)

The challenge we were facing was how do we define the distance between two 2D projection images and how this distance is connected to the distance between two quaternions. Since we are not able to create one with the desired invariants manually (e.g. invariant to translation), our goal was to learn a *distance metric* such that if the two 2D projection images are close in the embedding space of projections, the distance between the corresponding quaternions of rotation angles will be small in the quaternion space as well.

In order to do that, we use the Siamese Neural Network (SiameseNN)[10] to learn the distance metric between two 2D projection images. This is done only once in the beginning. The goal of SiameseNNs is to learn a similarity function which in our case is the distance between two 2D projection images d_p. With this network implementation we aim to classify (i.e. calculate the distance between two 2D projection images) the new unseen proteins without training the network again.

Ideally, we expect estimated distance d_p to be the function that is the equal to the function of the quaternion distance d_q. To learn the estimated distance $\hat{d}_p$ we use the SiameseNN with the following loss equation:

$$\hat{d}_p = \arg\min_{d_p} \sum_{i,j} |d_p(p_i, p_j) - d_q(q_i, q_j)|^2 \tag{5}$$

where the d_p is distance metric between two projection images p_i and p_j; and d_q is the quaternion distance between two quaternions q_i and q_j.

Using the software tools for data simulation we generate numerous 2D projection distances with their corresponding rotation angle values. Therefore, in the equation 19 we know the d_q, q_i, q_j as well as corresponding p_i, p_j and we are left to learn the best value of d_p.

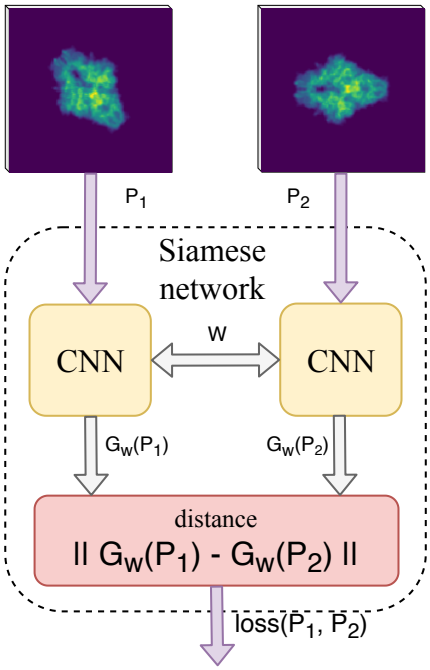

Figure 4: The architecture of the implemented Siamese Neural Network. The p_1 and p_2 are pairs of 2D projection images. Each one is the input to the corresponding CNN model. These two CNN models share the weights w. The output of two CNN models is a value of the function $G_w(p_1)$ and $G_w(p_2)$, respectively. These functions are embedding two inputs into some highly structured space, where this output is then be used by the following function calculating their element-wise absolute difference (see red box). The output of this Siamese Neural Network is a distance metric representing a distance between two input images p_1 and p_2 in an embedding space.

The SiameseNN is a type of a network that uses multiple instances of the same model and share same architecture and weights (the two sister convolutional neural networks (CNNs) on Fig.4). We have two copies of the same network that share the same weights. As input, we have two projection images (p_1 and p_2) which are passed through CNN models to generate a fixed-length feature vector for each ($G_w(p_1)$ and $G_w(p_2)$). If the two input projection images have similar features, then their feature vectors must also be similar, whereas if the two input projection images have different features, then their feature vectors will also be different. Thus the element-wise absolute difference between

the two feature vectors must be very different in both the above cases. Hence the loss value generated by the output loss layer must also be different in these two cases. This is the central idea behind the Siamese Networks[11].

2.4 Angle Recovery (Online)

Angle recovery is done every time we are provided with the new protein dataset for which we want to discover their corresponding angles. With this method we recover the angles. With this optimization we want to estimate projections' position in quaternion space assuming we know the distance metric between the projection images (done in distance estimation part, section 2.3).

With angle recovery we are embedding the projections into the quaternion space. The angle recovery represents the reverse method compared to the projection to graph conversion.

To estimate the quaternions (therefore the rotation angles), we optimize the following equation:

$$\{\hat{q}_i\}_{i=1}^{N} = \underset{\{\hat{q}_i\}_{i=1}^{N}}{\arg\min} \sum_{i,j} \left| d_p(p_i, p_j) - d_q(q_i, q_j) \right|^2 \tag{6}$$

where the d_p is distance metric between two projection images p_i and p_j; and d_q is the quaternion distance between two quaternions q_i and q_j.

In this case, the values of d_p, p_i, p_j as well as d_q are known and we are left to learn the best set of q_i values. Knowing the quaternion values, we are able to get the final values of the rotation angles using the conversion defined in the section 2.1.

In the beginning, we assumed we know the distance metric d_p by using the Euclidean distance as a baseline. With this we performed the angle recovery optimization just to confirm it starts to converge to lower loss.

We were not able to find some existing methods that would estimate the angles. One similar challenge, but with pose recovery in the Euclidean setting is described in paper [12].

3 Experiments

For the experiments, we use software tools designed for data generation thatrely on the ASTRA Toolbox[2] designed for 2D and 3D tomography that was used for accelerated simulated data generation.

The three main development phases of the project are:

1. Development of the **angle recovery** method;
2. Development of the **distance estimation** method;
3. **Performance characterization**.

In Phase 1 we use an Euclidean distance as estimated distance between the projections d_p (i.e. we assume we have a perfect distance metric). For the angle recovery itself, we try to recover the angles from the simulated projections of the protein that do not have any noise. In this step, we just want to make sure the optimization is doable.

In Phase 2 we implement the Siamese Neural Network in order to learn the perfect distance metric d_p. We run the angle recovery part with this learned distance metric in order to observe the results. In addition, we find the global rotation R that globally rotates all the estimated quaternions in order to minimize the angle recovery error.

Lastly, we planned to use simulated data with noise or real acquired images from the protein database[3].

3.1 Phase 1: Angle Recovery with the Perfect Distances

In this experiment we run the angle recovery assuming we have a perfect distance estimation d_p. Therefore, instead of $d_p(p_i, p_j)$ in the equation 6 we put $d_q(q_i, q_j)$ which represents the distance between the true quaternions. The optimization equation now looks like this:

$$\{\hat{q}_i\}_{i=1}^{N} = \underset{\{\hat{q}_i\}_{i=1}^{N}}{\arg\min} \sum_{i,j} \left| d_q(q_i, q_j) - d_q(\hat{q}_i, \hat{q}_j) \right|^2 \tag{7}$$

.

We start by random uniform distribution of the quaternion values $\hat{q}_i$ and $\hat{q}_j$ and we want to see if during the optimization we manage to estimate the true quaternions q_i and q_j.

For the optimization we use the tensorflow with the GPU support. We split the training data into batches of size 256 and we restrict ourselves to the half-spere of the asymmetric protein (called 5j0n). Other settings are:

- Number of steps: 100K
- Optimizer: Adam (gradient-based optimization)
- Learning rate: 0.01
- Number of projections: 5K
- Angle coverage: half-sphere
- Sampling: random

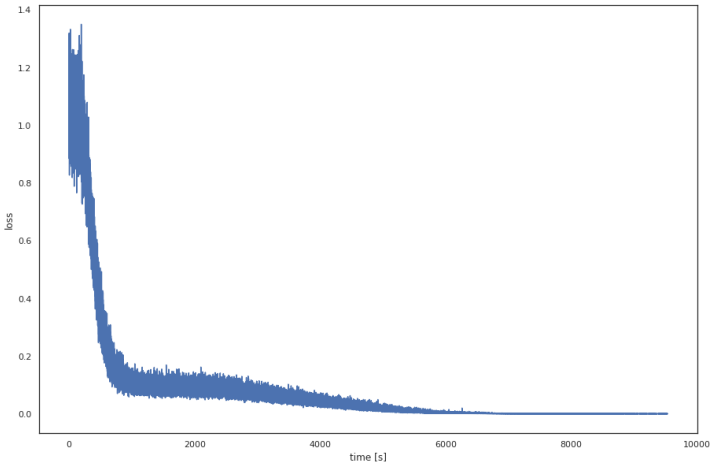

Figure 5: Optimization loss w.r.t. time

The optimization performace can be seen in Fig.5.

The optimization loss is 5.23e-04.

The sphere coverage in the Euler angle space can be seen in Fig.6 and represents the results of the angle recovery. We can see that the projection images cover the half sphere as expected. Together with the small loss value, we can confirm that the good performance of the algorithm is possible.

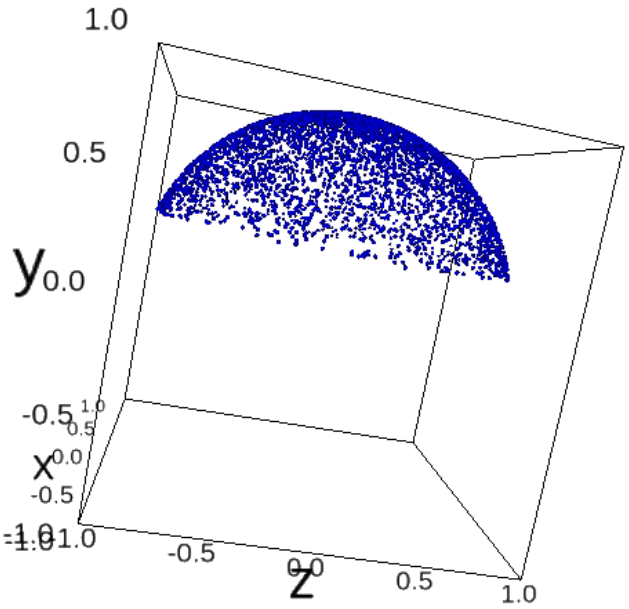

Figure 6: The sphere coverage after the angle recovery optimization in where the d_p is assumed to be the quaternion distance between corresponding quaternions.

This optimization equation 7 is used to measure our success (i.e. the ground truth loss) in the following phases.

[2]https://github.com/astra-toolbox/astra-toolbox

[3]https://www.rcsb.org/

In addition, the main problem is that we can not just take the error between the estimated and true angles since they are not invariant to rotation (i.e. any global rotation of angles is as valid as any other). Hence, here we concentrate on computing the error on the distances.

In the end of this phase, we can conclude that it is possible to recover the angles from **distances** using the perfect distance estimation.

3.2 Phase 2.1: Angle Recovery with Estimated Distances

In this experiment we run the angle recovery using the baseline Euclidean distance as our distance metric d_p. Therefore, the value of d_p is $d_p(p_i, p_j) = \sqrt{\sum_k^n (p_{i_k} - p_{j_k})^2}$ in the equation 6. With this experiment we want to see if there is a linear dependence between d_p and d_q, and is an Euclidean distance for d_p a good estimation for d_q: $d_p(p_i, p_j) = C \cdot d_q(q_i, q_j)$, where C is a constant.

In this experiment, we do not use the ground truth angles in the optimization. We want to estimate the angles only using the projections and their estimated distance (in this case Euclidean distance).

From the plot in Fig.7 we cna see that smallest distances tend to be linear. This can happen due to the symmetries in the proteins or projections. It seems that the Euclidean distance is predictive for small distances, but flat afterwards.

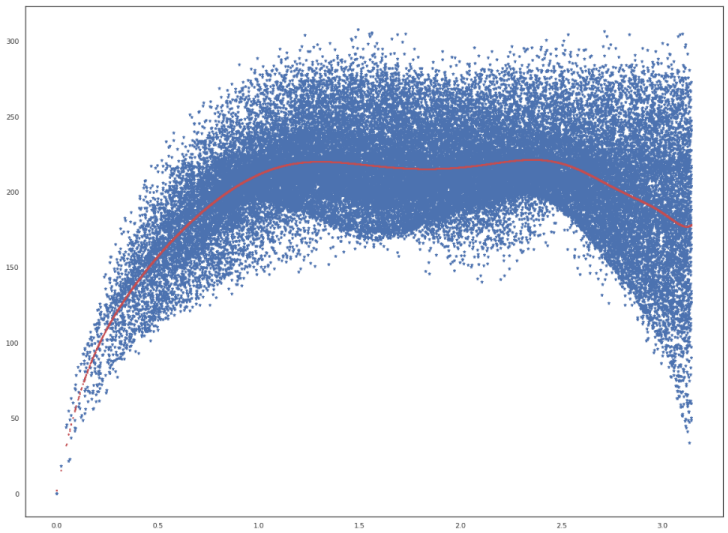

Figure 7: The d_q values on the x-axis and d_p values on the y-axis. The plot resents the ratio $\frac{d_p}{d_q}$ of distances between 10 random projections compared to all the other projections for each.

In the plot in Fig.8 we can observe that there exists the linear relation between d_p and d_q. This confirms our observation from Fig.7.

In the end of this phase, we can conclude that the baseline Euclidean distance for d_p is a good estimate. However, only for the small distances in the simplest case (i.e. without noise, shift, etc.).

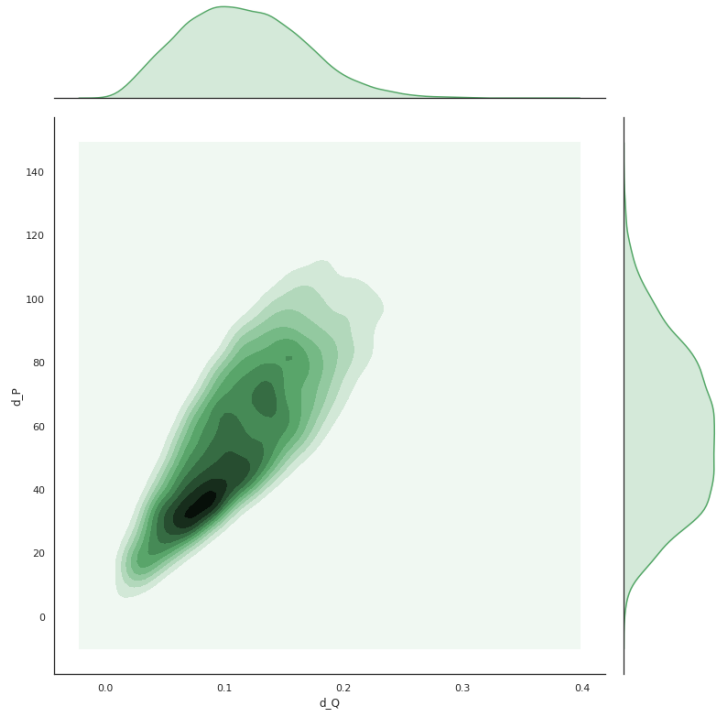

Figure 8: The d_q values on the x-axis and d_p values on the y-axis. The plot resents the ratio $\frac{d_p}{d_q}$ of 5 closest distance pairs calculated for every projection image.

Testing different penalizations for small and large distances The next step for this phase was to run the angle recovery with the assumption of d_p being the Euclidean distance between two projection images. The settings were following:

- Number of steps: 10K
- Optimizer: Adam (gradient-based optimization)
- Learning rate: 0.001
- Number of projections: 5K
- Angle coverage: half-sphere
- Sampling: half closest knn projection image pairs + half random projection image pairs

The result we got is the loss 2.20 and the sphere coverage can be seen in Fig.9. We can observe that the projection images in Euler angle space are concentrating in one place. This concentration happens due to the large distances not being well estimated.

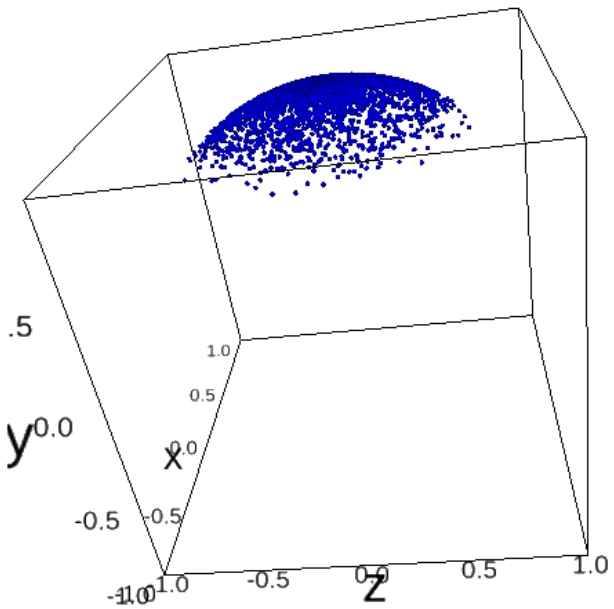

Figure 9: The sphere coverage after the angle recovery optimization in where the d_p is assumed to be the Euclidean distance.

3.3 Phase 2.2: Distance Estimation with the Siamese Neural Network

In this experiment we train the Siamese Neural Network to find the perfect distance metric d_p. We expect to have a linear dependence between d_p and d_q outputs (i.e. $d_p(p_i, p_j) = C \cdot d_q(q_i, q_j)$). Ideally, we expect the input parameters of the d_p method to be two projection images and the output to be be their distance in the quaternion space ($C = 1$).

The architecture of implemented Siamese Neural Network can be seen in Fig.10. We can see the two twin sub-networks with the same layers on the left and the right side of the figure. The left side of the network accepts the first element of the distance pair as an input and the right side accepts the second element of the distance pair as an input. The last layer unites these two sub-networks and outputs the single value that represents the Euclidean distance between the feature vectors received for each of the sub-network. The output of the last layer (i.e. the Euclidean layer) represents the similarity measure between the two projection images. In our case, this similarity measure represents the distance in the embedding space between two input projection images.

The settings for training the SiameseNN were: 500 epochs with 60K training distance pairs, 20K test distance pairs, and 20K validation distance pairs. Training, test, and validation sets were creating using the disjoint sets of projection images (i.e. since we have 5K projection images, 3K projection images were used to create 60K training distance pairs, 1K projection images were used for test and validation sets each). The performance of SiameseNN training can be seen in Fig.11.

The ratio between d_p and d_q can be seen in Fig.12. We can see that the SiameseNN managed to learn the linear dependence with $C = 1$.

After seeing that the SiameseNN performs quite good, we continued with the angle recovery optimization. Before the angle recovery, the loss between ground truth angles

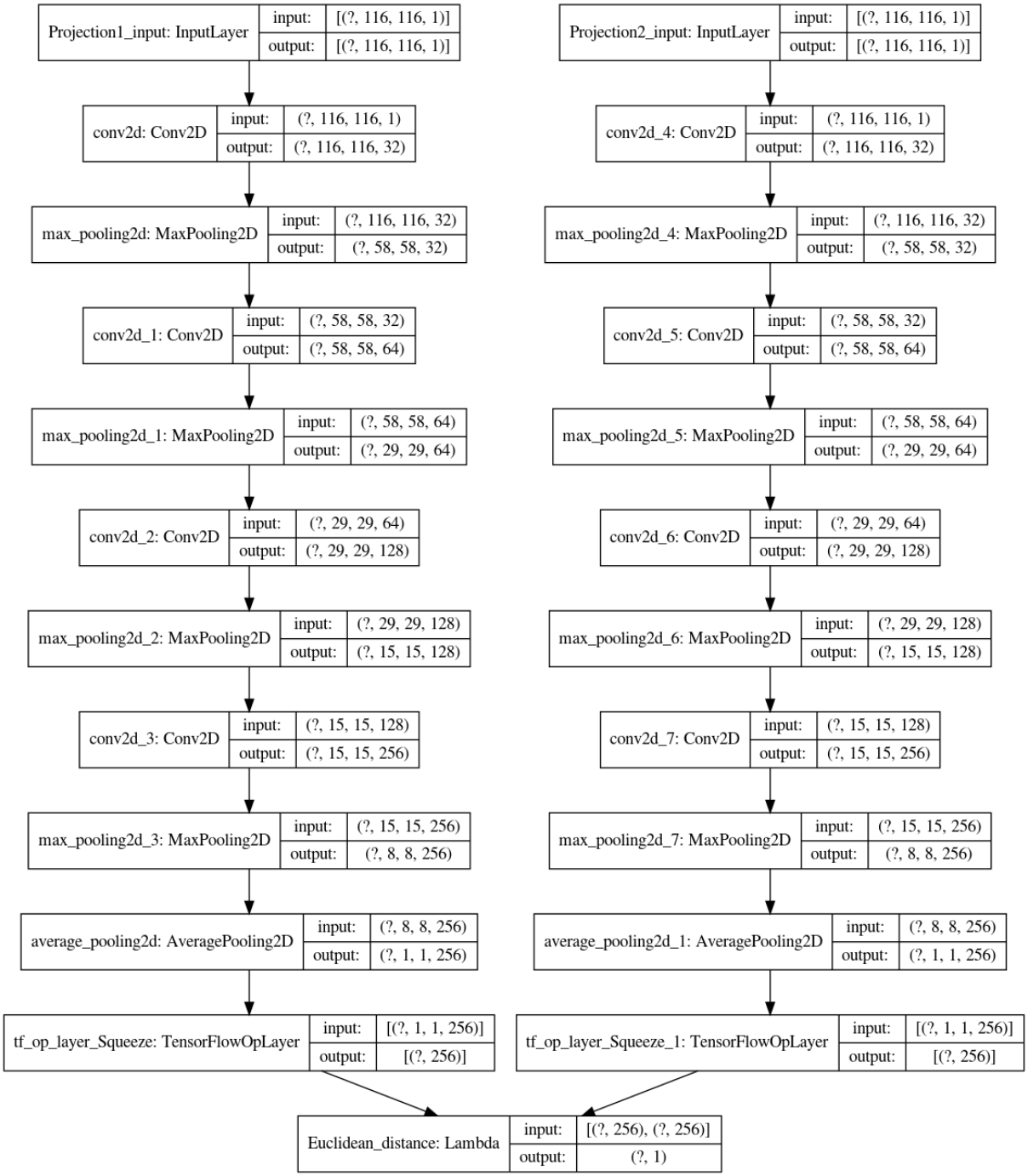

Figure 10: Architecture of the Siamese Neural Network.

and the estimated ones is 1.097 (calculated using the equation 7). The loss for angle recovery using the output of SiameseNN as distance metric d_p was 8.37e-01. The loss between ground truth angles and the estimated ones is 1.0054.

The sphere coverage of 10 (true vs. predicted) projection images' embeddings can be seen in Fig.13. We can see that the angle recovery is not performing as well as expected. The possible areas of improvement could be the modification of the SiameseNN's model layers as well as modifying the loss function for the angle recovery.

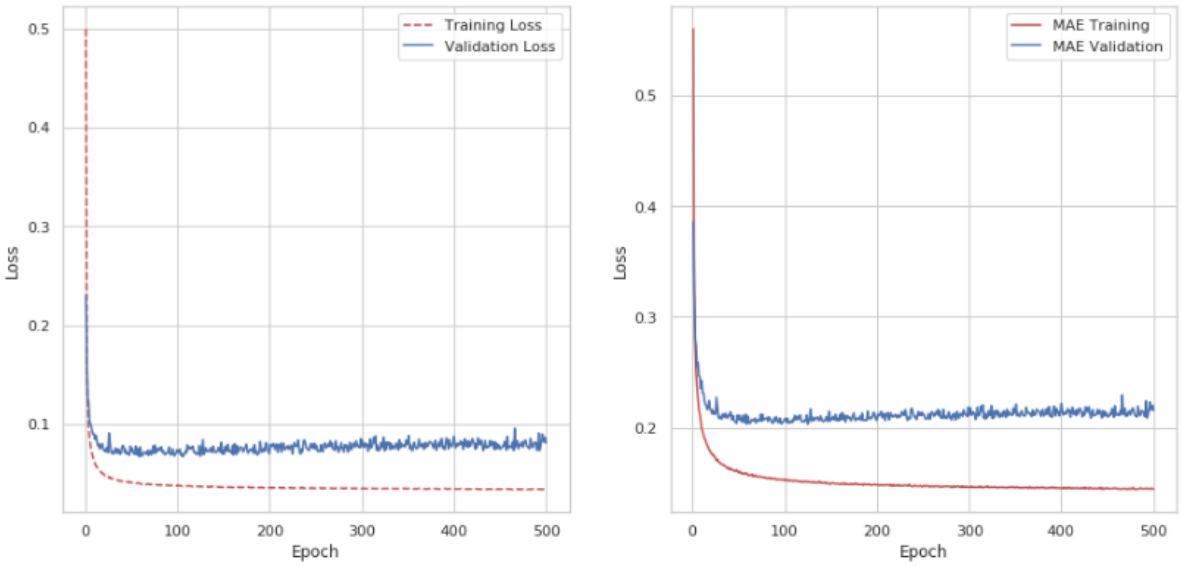

Figure 11: Training and validation losses of SiameseNN. Left plot shows the MSE and the right plot shows the MAE error w.r.t. epoch.

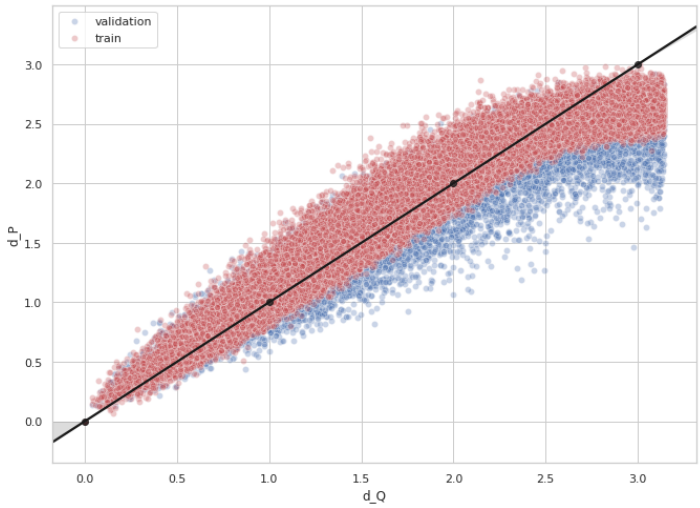

Figure 12: Ratio between d_p and d_q. The red dots represent the training set and blue dots represent validation data. The back line is the line $f(x) = x$.

Angle Estimation Error Metric In order to better understand how far is the estimated result from the optimal result, we are introducing the mean average angle estimation error under the best global alignment:

$$\hat{R} = \arg\min_{R} \frac{1}{N} \sum_{i=1}^{N} |d_q(q_i, R\hat{q}_i)| \tag{8}$$

where d_q is the quaternion distance between q_i and $R\hat{q}_i$; q_i is the true quaternion; $\hat{q}_i$ is estimated quaternion; R is a quaternion representing a global rotation which best aligns the two sets of quaternions. The estimated quaternions are globally rotated as $R\hat{q}_i$ where the multiplication of the two quaternions is the composition of the two rotations they represent.

The general rotation R would rotate all the estimated quaternions in order to minimize

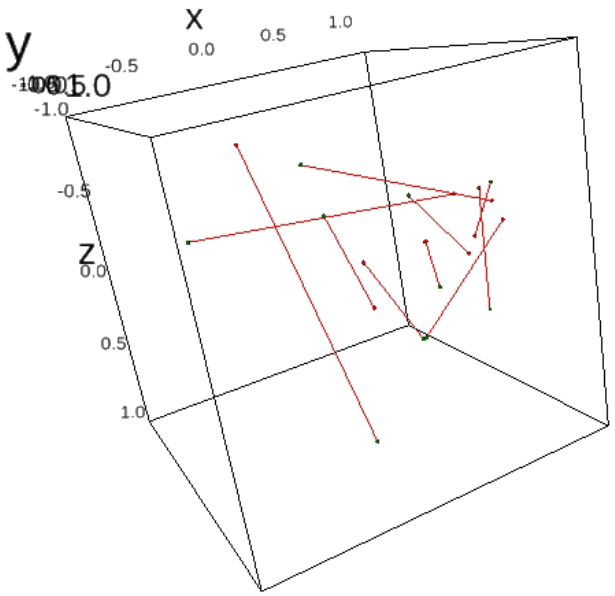

Figure 13: The sphere coverage of 10 predicted vs. true projection image embeddings in the Euler angle space. The red dots are true angles and the green dots are the estimated angles. The lines connecting them are their distances

the average distance between the true quaternions and estimated quaternions rotated using the rotation quaternion R.

The initial rotation around $Z-Y-Z$ axes are the following three rotation angles (0, 0, 0). The initial loss value is 1.871 calculated using the equation 8. After running the optimization, the values of rotation angles are (1.48, 0.006, 1.36) and the calculated loss is 1.86. In other words, this is around 106.5 degrees of difference between the estimated rotated angles' distances and the true ones. This value is very high if we take into the account the fact that we would like the accept the solution that has difference at most 10 degrees.

Angle recovery with true angles initialization Previously, we initialized our estimated angles with random uniformly distributed angle values in every axis $Z-Y-Z$. We wondered how would the angle recovery perform depending on the initialization of estimated angles. We put them to be the true (correct) angles.

initialization	GT loss before	AR loss	GT loss after	R loss before (deg)	R loss after (deg)	GT loss final
random angles	1.0971	8.37e-01	1.005	1.871 (107.2°)	1.867 (106.5°)	1.0075
true angles	0.0	8.30e-01	0.324	0.186 (10.66°)	0.188 (10.77°)	0.324

Table 1: Losses depending on the initialization of estimated angles; GT loss = ground truth loss (calculated with the equation 7); R loss = rotation loss (calculated with the equation 8); AR loss = angle recovery loss (calculated with the equation 6).

In Table 1 we can see the performance of angle recovery with true angles compared to the random angle initialization. The ground truth loss after the angle recovery decreased when the estimated angles are initialized randomly. However, if they are initialized with

true angles, we can see the increase in loss after the angle recovery (by 0.32). The rotation loss decreases by 0.7°. The ground truth loss increases slightly (0.002) but it is still better than it was in the beginning (1.097). The rotation loss in true angles initialization slightly increases after the global rotation (by 0.1°). Even so, the rotation loss is very good (10°) with the true angle initialization.

We are still left with some crucial questions like: Why the angle recovery loss seems to be very similar in both cases? Why does ground truth loss increase after the angle recovery in case of true angle initialization? We hope to answer them in the work that will follow.

4 Discussion

Overall, the main difficulty of our project presently is to recover the angles with an approximate distance estimation. We have three scenarios:

- Using the true distance as a distance metric d_p (see section 3.1). We saw that the angle recovery works perfectly. Knowing this, we concluded that finding the perfect distance metric using the Siamese Neural Network has the high potential to work.
- Using the Euclidean distance as a distance metric d_p (see section 3.2). We saw in Fig.7 and Fig.8 that the Euclidean distance is only good for small distances. After testing different penalizations (see section 3.2) we could not make the angle recovery work.
- Using the learned distance metric d_p. This distance metric was learned using the Siamese Neural Network (see section 3.3). We managed to learn the (almost) perfect distance metric. It performs well (the optimization losses decrease), however the angle estimation is still noisy. We can say that it partially works and there is still place for improvements.

In this project we worked with the two proteins called 5a1a[4] (protein with symmetries) and 5j0n[5] (asymmetric protein).

4.1 Future Work

The stages of the project development are the following:

1. Testing the **feasibility** of the algorithms as well as the whole pipeline. In this stage we are only using one protein 3D volume $\mathbf{x}$. The set of projections is created using the projection $\mathbf{P}_{\theta_i}$ along the 3D pose θ_i. We do not consider the effects of noise, shift, etc. (see the full equation 1). The 2D projection set $\{\mathbf{y}_i\}$ is then split into disjoint training $\{\mathbf{y}_i^{train}\}$ and test $\{\mathbf{y}_i^{test}\}$ sets.

   $$\mathbf{x} \rightarrow \{\mathbf{y}_i\} = \{\mathbf{P}_{\theta_i}\mathbf{x}\} \implies \{\mathbf{y}_i^{train}\} \cup \{\mathbf{y}_i^{test}\} = \{\mathbf{y}_i\} \wedge \{\mathbf{y}_i^{train}\} \cap \{\mathbf{y}_i^{test}\} = \emptyset$$

2. Testing the **robustness to noise n**. This stage presents the extension of the previous stage. Same as before, we are only using one protein 3D volume $\mathbf{x}$ and the set of projections is created using the projection $\mathbf{P}_{\theta_i}$ along the 3D pose θ_i. Here we include the noise $\mathbf{n}$ to crate the set of 2D projections $\{\mathbf{y}_i\}$. The 2D projection set $\{\mathbf{y}_i\}$ is then split into disjoint training $\{\mathbf{y}_i^{train}\}$ and test $\{\mathbf{y}_i^{test}\}$ sets.

   $$\mathbf{x} \rightarrow \{\mathbf{y}_i\} = \{\mathbf{P}_{\theta_i}\mathbf{x} + \mathbf{n}\} \implies \{\mathbf{y}_i^{train}\} \cup \{\mathbf{y}_i^{test}\} = \{\mathbf{y}_i\} \wedge \{\mathbf{y}_i^{train}\} \cap \{\mathbf{y}_i^{test}\} = \emptyset$$

3. Testing the **robustness to unseen protein volumes**. In this stage, we have N protein 3D volumes that are used to create the training set of 2D projections $\{\mathbf{y}_{N,i}^{train}\}$. Another, unseen protein 3D volume $\mathbf{x}^{test}$ is used to create the test set of 2D projections $\{\mathbf{y}_i^{test}\}$. The set of projections is created using the projection $\mathbf{P}_{\theta_i}$ along the 3D pose θ_i. We do not consider the effects of noise, shift, etc.

   $$\{\mathbf{x}_1^{train}, ..., \mathbf{x}_N^{train}\} \rightarrow \{\mathbf{y}_{N,i}^{train}\} = \{\mathbf{P}_{\theta_i}\mathbf{x}_N^{train}\}$$

   $$\mathbf{x}^{test} \rightarrow \{\mathbf{y}_i^{test}\} = \{\mathbf{P}_{\theta_i}\mathbf{x}^{test}\}$$

[4] https://www.rcsb.org/structure/5A1A
[5] https://www.rcsb.org/structure/5J0N

4. Testing the **faithfulness** of $\mathbf{H}_\phi$. We are only using one protein 3D volume $\mathbf{x}$ and the set of projections is created using the projection $\mathbf{P}_{\theta_i}$ along the 3D pose θ_i as well as all the other operators (noise $\mathbf{n}$, CTF $\mathbf{C}_\varphi$, shift $\mathbf{S_t}$) specified in the equation 1. The 2D projection set $\{\mathbf{y}_i\}$ is then split into disjoint training $\{\mathbf{y}_i^{train}\}$ and test $\{\mathbf{y}_i^{test}\}$ sets.

   $\mathbf{x} \rightarrow \{\mathbf{y}_i\} = \{\mathbf{C_\varphi S_t P}_{\theta_i}\mathbf{x} + \mathbf{n}\} = \{\mathbf{H}_\phi\mathbf{x} + \mathbf{n}\} \implies \{\mathbf{y}_i^{train}\} \cup \{\mathbf{y}_i^{test}\} = \{\mathbf{y}_i\} \wedge \{\mathbf{y}_i^{train}\} \cap \{\mathbf{y}_i^{test}\} = \emptyset$

5. Final **goal**. We have N protein 3D volumes that are used to create the training set of 2D projections $\{\mathbf{y}_{N,i}^{train}\}$. Another, unseen protein 3D volume $\mathbf{x}^{test}$ is used to create the test set of 2D projections $\{\mathbf{y}_i^{test}\}$. The set of projections is created using the projection $\mathbf{P}_{\theta_i}$ along the 3D pose θ_i as well as all the other operators (noise $\mathbf{n}$, CTF $\mathbf{C}_\varphi$, shift $\mathbf{S_t}$) specified in the equation 1.

   $\{\mathbf{x}_1^{train}, ..., \mathbf{x}_N^{train}\} \rightarrow \{\mathbf{y}_{N,i}^{train}\} = \{\mathbf{C_\varphi S_t P}_{\theta_i}\mathbf{x}_N^{train} + \mathbf{n}\}$

   $\mathbf{x}^{test} \rightarrow \{\mathbf{y}_i^{test}\} = \{\mathbf{C_\varphi S_t P}_{\theta_i}\mathbf{x}^{test} + \mathbf{n}\}$

In this project we only covered the stage 1. Other stages can be considered to be the future work. In addition to these stages, the following things could be improved:

- Improve the Siamese Neural Network model by adding more complexity (i.e. modifying the existing layers and adding the new layers). The performance can be compared using the results seen in the Fig.11. The goal would be to decrease the difference between training and validation losses.

- The current implementation of Siamese Neural Network unites the two twin CNN networks using the predefined metric - Euclidean distance metric. We might consider implementing the learned metrics - nonlinear distance metric.

- Add the last step done after the angle recovery, called protein reconstruction. The protein reconstruction uses already developed advanced algorithms that reconstructs the 3D volume given the protein 2D projection images and their corresponding rotation angles.

- Run the angle recovery with estimated angles that are initialized with true angles + 5°, 10°, etc. difference. Observe the results and compare the performance with random angles initialization.

5 Conclusion

The challenge of the project was to learn a pairwise projection distances in order to recover the angles at which the 2D projection of the 3D protein volume was taken from.

Exploring different results, we learned how different assumptions and setups affect the performance of the pipeline. The first biggest challenges was to find the perfect distance metric between two projection images. The latest implementation of the pipeline includes learning the distance metric using the Siamese Neural Network since we are not able to create the distance metric with the desired invariants manually. The second biggest challenge was to recover the angles from these learned distances.

In the end, we successfully managed to run the pipeline and observe the positive results regarding the feasibility of the proposed approaches. However, there is a lot of space for future improvements in the pipeline.

References

[1] J. Dubochet, "Cryo-electron microscopy of vitrified specimens," *Quarterly Reviews of Biophysics*, vol. 21, no. 2, pp. 129–228, 1988.

[2] "Cryogenics | physics | Britannica." [Online]. Available: https://www.britannica.com/science/cryogenics

[3] L. Donati, M. Nilchian, C. O. S. Sorzano, and M. Unser, "Fast multiscale reconstruction for Cryo-EM," *Journal of Structural Biology*, vol. 204, no. 3, pp. 543–554, Dec. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1047847718302697

[4] "Greg Pintilie." [Online]. Available: https://people.csail.mit.edu/gdp/cryoem.html

[5] F. DiMaio, D. A. Kondrashov, E. Bitto, A. Soni, C. A. Bingman, G. N. Phillips, and J. W. Shavlik, "Creating protein models from electron-density maps using particle-filtering methods," *Bioinformatics*, vol. 23, no. 21, pp. 2851–2858, Nov. 2007. [Online]. Available: https://academic.oup.com/bioinformatics/article/23/21/2851/374177

[6] B. A. Rosenfeld, *A History of Non-Euclidean Geometry: Evolution of the Concept of a Geometric Space*. Springer New York, Sep. 1988, google-Books-ID: DRLpAFZM7uwC.

[7] "Maths -Quaternion Transforms - Martin Baker." [Online]. Available: http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm

[8] W. R. Hamilton, "On quaternions, or on a new system of imaginaries in algebra," p. 92.

[9] D. Q. Huynh, "Metrics for 3d Rotations: Comparison and Analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, Oct. 2009. [Online]. Available: https://doi.org/10.1007/s10851-009-0161-2

[10] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a Similarity Metric Discriminatively, with Application to Face Verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. San Diego, CA, USA: IEEE, 2005, pp. 539–546. [Online]. Available: http://ieeexplore.ieee.org/document/1467314/

[11] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition," p. 8.

[12] I. Dokmanic, R. Parhizkar, J. Ranieri, and M. Vetterli, "Euclidean distance matrices: essential theory, algorithms, and applications," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 12–30, 2015.

[13] "Orientation, Rotation, Velocity, and Acceleration and the SRM," p. 69.

6 Appendix

The conversion depends on the sequence of rotation axes. There exist twelve possible sequences of rotation axes, divided in two groups: (1) proper Euler angles and (2) Tait-Bryan angles[6].

In our project, we are dealing with the sequence of rotation axes $Z-Y-Z$ that belongs to the proper Euler angles group. The TensorFlow[7] package implements the quaternions class. However, the quaternion-Euler angle conversion only supports the Tait-Bryan angles (sequence of rotations around $Z-Y-X$ axes). Thus we implemented the conversion in TensorFlow that supports our sequence of the rotation axes.

Euler angle to quaternion conversion Let's assume that α, β, and γ are the rotation angles around the $Z-Y-Z$ axes respectively. In order to convert it to the quaternion, we use following equations:

$$c_1 = cos(\frac{\alpha}{2}); c_2 = cos(\frac{\beta}{2}); c_3 = cos(\frac{\gamma}{2}); \tag{9}$$

$$s_1 = sin(\frac{\alpha}{2}); s_2 = sin(\frac{\beta}{2}); s_3 = sin(\frac{\gamma}{2}) \tag{10}$$

$$w = c_1c_2c_3 - s_1c_2s_3 \tag{11}$$

$$x = c_1s_2s_3 - s_1s_2c_3 \tag{12}$$

$$y = c_1s_2c_3 + s_1s_2s_3 \tag{13}$$

$$z = c_1c_2s_3 + s_1c_2c_3 \tag{14}$$

. Calculating the w, x, y, and z values and incorporating them in the equation 2, we get the quaternion representation.

Quaternion to Euler angle conversion Let's assume that we have the w, x, y, and z parts of the quaternion. In order to convert it to the rotation angles around the $Z-Y-Z$ axes, we use the following equations:

$$R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz+wx) \\ 2(xz+wy) & 2(yz-wx) & 1-2(x^2+y^2) \end{bmatrix} \tag{15}$$

To factor $M = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$ that belongs to SO(3)[8] into a sequaence of principal factors in the Euler angle $Z-Y-Z$ rotation convention $R_Z(\alpha)R_Y(\beta)R_Z(\gamma)$, we expand the sequence by multiplying the corresponding matrix forms (equation 1.15 in [13]). The resulting matrix is:

[6]https://en.wikipedia.org/wiki/Euler_angles

[7]https://www.tensorflow.org/

[8]https://en.wikipedia.org/wiki/3D_rotation_group

$$R_Z(\alpha)R_Y(\beta)R_Z(\gamma) =$$
$$\begin{bmatrix} cos(\beta)cos(\alpha)cos(\gamma) - sin(\alpha)sin(\gamma) & -cos(\gamma)sin(\alpha) - cos(\beta)cos(\alpha)sin(\gamma) & sin(\beta)cos(\alpha) \\ cos(\beta)cos(\gamma)sin(\alpha) + cos(\alpha)sin(\gamma) & cos(\alpha)cos(\gamma) - cos(\beta)sin(\alpha)sin(\gamma) & sin(\beta)sin(\alpha) \\ -sin(\beta)cos(\gamma) & sin(\beta)sin(\gamma) & cos(\beta) \end{bmatrix}$$

Matching the elements of this matrix $R_Z(\alpha)R_Y(\beta)R_Z(\gamma)$ to those of M, we find that:

$$a_{22} = cos(\beta) \tag{16}$$
$$\frac{a_{21}}{(-a_{20})} = tan(\gamma) \tag{17}$$
$$\frac{a_{12}}{a_{02}} = tan(\alpha) \tag{18}$$

We also need to take the gimbal lock into an account [13]. The gimbal lock refers to a gyroscope having three nested gimbals to provide three degrees of rotational freedom. There exist critical angles for the middle gimbal that reduces the rotational degrees of freedom from three to two. In these critical configurations, the gimbals lie in a single plane and rotation in that plane is locked out by gimble mechanism. This loss of degree is a *gimbal lock*. In the Table 2, all three degrees of freedom are available when $\beta \neq 0$ and $\beta \neq \pi$ (i.e. $a_{22} \neq \pm 1$). We have only two degrees of freedom when $\beta = 0$ or $\beta = \pi$ (i.e. $a_{22} = -1$ or $a_{22} = +1$). The consecutive rotations collapse down to a single principal rotation:

$$\beta = 0 : R_Z(\alpha)R_Y(0)R_Z(\gamma) = R_Z(\alpha)R_Z(\gamma) = R_Z(\alpha + \gamma) \tag{19}$$
$$\beta = \pi : R_Z(\alpha)R_Y(\pi)R_Z(\gamma) = R_Z(\alpha)R_Z(-\gamma) = R_Z(\alpha - \gamma) \tag{20}$$

All these equations lead to the solution in Table 2 based on value a_{22}:

Case	Principal factors for rotation $R_Z(\alpha)R_Y(\beta)R_Z(\gamma)$ (all angles modulo 2π)		
$a_{22} \neq \pm 1$	$\beta = arccos(a_{22})$ [principal value] $0 < \beta < \pi$	$\alpha = arctan2(a_{12}, a_{02})$	$\gamma = arctan2(a_{21}, -a_{20})$
	$\beta = arccos(a_{22})$ [2π - principal value] $\pi < \beta < 2\pi$	$\alpha = arctan2(-a_{12}, -a_{02})$	$\gamma = arctan2(-a_{21}, a_{20})$
$a_{22} = -1$	$\beta = \pi$	$\alpha = arctan2(a_{10}, a_{11}) + \gamma$	any value of γ
$a_{22} = +1$	$\beta = 0$	$\alpha = arctan2(a_{10}, a_{11}) - \gamma$	any value of α

Table 2: Principal factors for $Z - Y - Z$ rotation.

Calculation of the angles α, β, and γ above was made possible thanks to the paper [13].