# Deep Learning Project Final Report
# Kaggle AI Challenge
# CryoET Object Identification

Fatemeh Gol Pour and Anna Karolin

January 24, 2025

**Abstract**

Cryoelectron tomography (cryoET) enables capturing high-resolution 3D images of protein complexes, which are essential for cellular function and health. The automatic identification and annotation of these protein complexes in cryoET tomograms is a significant challenge. This report represents our approaches to solving this problem by applying deep learning models to segment and classify protein complexes within cryoET tomograms. We evaluated two architectures(3D U-NET and VNET) on a cryoET dataset consisting of six classes of protein complexes.

## 1 Introduction

Proteins, such as oxygen-carrying hemoglobin or keratin in hair, are essential for cell function, and understanding their interactions is crucial for human health and disease treatments. Cryo-electron tomography (cryoET) generates 3D images, known as tomograms, that provide near-atomic resolution, revealing proteins in their natural, complex, and crowded environments. This technique holds immense potential for uncovering the mysteries of the cell.

Despite the quality of cryoET tomograms available, much of this data remains untouched. The cryoET data portal (`https://cryoetdataportal.czscience.com`) hosts a growing collection of published datasets in a standardized format. To make sense of this data, the automatic identification of protein molecules within these tomograms is needed. Even proteins that are distinguishable by the human eye pose challenges for automated identification. Developing a generalizable solution for this problem will reveal the dark matter of the cell and contribute to numerous breakthroughs in human health.

The kaggle challenge CryoET Object Identification was proposed to us as our Deep Learning II project. It is a supervised learning problem which aims to create deep learning algorithms capable of automatically annotating five classes of protein complexes within a real-world cryoET dataset.

## 2 Dataset

The dataset consists of cryoET tomograms, which are 3D images of protein complexes. These tomograms are represented by voxels in 3D space, with each object type identified and labeled.

The dataset contains several classes of protein complexes, and each class of object has a corresponding radius, which helps in defining its size and the region in which it resides within the tomogram:

- ribosome (radius: 150 nm, or 15 voxels)
- virus-like particles (radius: 135 nm, or 13.5 voxels)
- apo-ferritin (radius: 60 nm, or 6 voxels)
- thyroglobulin (radius: 130 nm, or 13 voxels)
- $\beta$-galactosidase (radius: 90 nm, or 9 voxels)
- $\beta$-amylase

The train folder consists of:

- Static Data (Inputs) containing seven 3D tomograms in Zarr format. Each experiment includes wbp (raw data), isonetcorrected (data that isn't blurred), denoised, and ctfdeconvolved subfolders, from which we used the denoised data, and the highest resolution (0 from the available options: 0, 1, and 2). Each voxel represents a 10nm10nm10nm cube.
- Overlay Data (Labels) with JSON files containing particle centroids and object types.

And the test folder contains the input images, with no labels provided.

Figure 2 shows a ribosome particle in a 2D slice along the z axis in the TS_6_2 tomogram, while figure 1 is the 3D representation of all the different particles in TS_6_4. Additionally, figure 3 indicates the imbalance of classes along different experiments(tomograms).

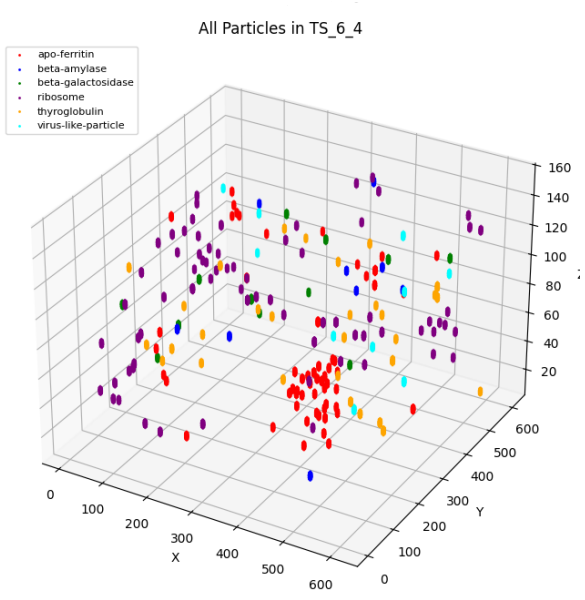


Figure 1: 3D distribution of the particles

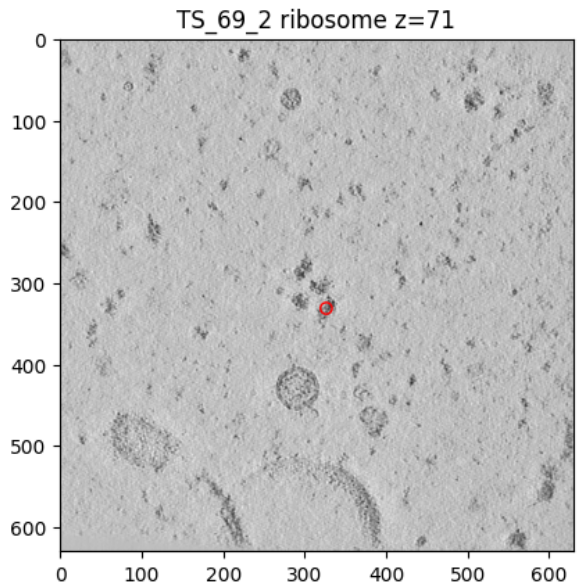


Figure 2: 2D slice showing ribosome

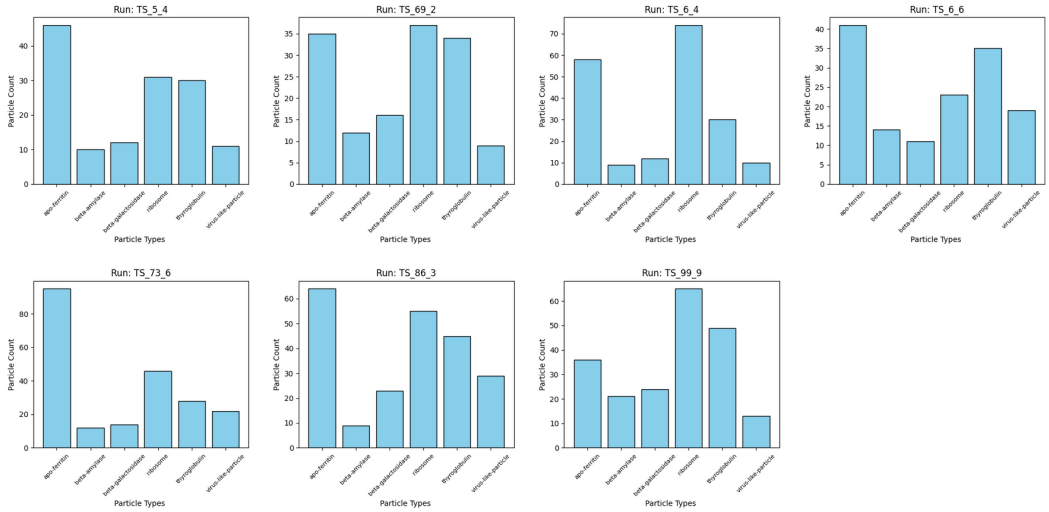


Figure 3: Frequencies of different particles in different tomograms

## 2.1 Preprocessing

The data is given in the form of seven tomograms (experiments). We initialize lists to store the filtered volumes and labels for all experiments. For each experiment, we load the Zarr dataset and annotation data (centroids) associated with the run. Then we map the centroids to a 3D voxel grid and create a tensor for the voxel labels.These voxel labels are into spherical regions (blobs) to simulate the spread of objects like ribosomes in 3D space, creating more continuous regions. then we extract sub-volumes of the data as smaller 3D cubes, where we check for labeled objects (positive samples) and background (negative samples) and normalize and store the filtered sub-volumes and labels in tensors. After processing all experiments, we concatenate the filtered volumes and labels into larger datasets. The data is then warped in PyTorchs TensorDataset to create a dataset ready for training. We split the dataset into training, validation, and test sets using the specified ratios and create DataLoader objects for each dataset, enabling batch processing during model training. This process is implemented using the following functions:

- read_zarr:
  The read_zarr function loads a .zarr dataset by taking a string run that specifies the experiment. It uses zarr.open() to open the file from a predefined directory and returns the dataset, which is organized hierarchically.

- read_run:
  The read_run function loads annotation data in .json format by searching for files related to the specified experiment run and combining them into a single pandas data frame. It then extracts the 3D coordinates from the "location" field, resulting in a unified data frame with annotated points' coordinates.

- map_overlay_to_voxels:
  The map_overlay_to_voxels function converts annotated points from a data frame into a 3D voxel grid by calculating voxel indices based on the point's coordinates and a voxel_spacing factor. It sets the voxel value to 1 for ribosomes and 0 for other points.

- expand_labels_with_blobs:
  The expand_labels_with_blobs function expands labeled regions in the voxel grid by creating spherical "blobs" around each labeled point using a specified radius. It identifies centroids and extends the labels into surrounding regions, simulating the spread of objects in 3D space.

- preprocess_and_filter_subvolumes:
  The preprocess_and_filter_subvolumes function extracts smaller 3D sub-volumes from a larger volume, checking for labeled objects and categorizing them as "positive" or "background" sub-volumes. It can sample background sub-volumes to balance the dataset and normalize the data, returning preprocessed sub-volumes and labels for machine learning tasks.

# 3 Evaluation Metrics

Below are the metrics used to evaluate our models.

Notations: $TP$ (True Positives) are the number of correctly predicted particle labels. $TN$ (True Negatives) are the number of correctly predicted background labels. $FP$ (False Positives) are the number of labels that are predicted as particles while actually being background. $FN$ (False Negatives) are the number of incorrectly predicted background labels.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

We consider recall to be the most prominent evaluation metric in this work, because it will evaluate our model's ability to identify as many relevant objects as possible.

# 4 Model 1 : Using 3D U-NET

## 4.1 Baseline Model

Initially, we trained a very simple baseline model on a single tomogram for (TS_69_2) to predict the ribosome particle. The architecture included:

- Encoder:
  - A 3D convolutional layer with 1 input and 32 outputs
  - ReLU activation function: introducing non-linearity
  - MaxPool3d: downsampling to reduce spatial dimension
- Decoder:
  - A 3D convolutional layer to reduce the feature mapsback to a single channel
  - Sigmoid activation function for binary segmentation

- Upsampling: To restore the resolution of the input data

But this baseline lacked a proper architecture, was not complex enough for the amount of data we have, and we were using BCELoss which is not a compatible loss function for an image segmentation task. As a result, the model performed poorly. The evaluation results were: Precision: 58.94%, Recall: 21.31%, and F1 Score: 31.31%. To improve these results, we decided to leverage a 3D UNET architecture.

## 4.2 3D U-NET Model Final Architecture

This architecture is an encoder-decoder network that uses 3D convolutions for segmentation. The encoder extracts features at multiple levels of abstraction, while the decoder reconstructs the segmented image. Skip connections are used to preserve spatial resolution. This model is well-suited for volumetric segmentation tasks like cryoET image analysis.

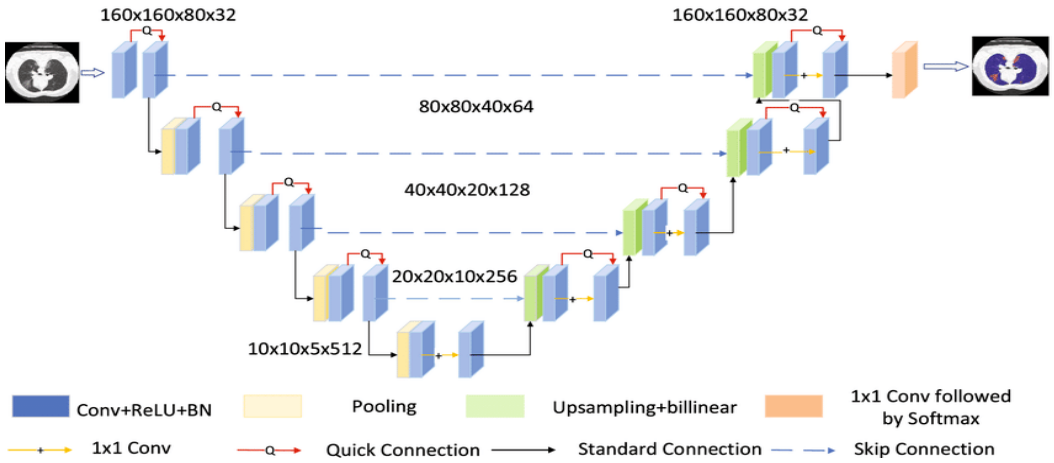


Figure 4: Example 3D U-NET Architecture `https://www.researchgate.net/figure/3D-U-Net-architecture-diagram_fig4_356508313`

In our work, we used a simplified version of the architecture shown in figure 4, with fewer layers. Here is a breakdown of our architecture:

- Contracting Path (Encoder): This part of the network is designed to capture the context of the input data by progressively downsampling it.
  - enc1 (First Encoding Block):
    - Conv3d(1, 32, kernel_size=3, padding=1): The first convolutional layer takes a 3D input and applies 32 filters with a kernel size of 3x3x3. Padding of 1 ensures that the spatial dimensions remain the same after the convolution.
    - BatchNorm3d(32): Normalizes the output of the convolution, which helps with training stability.
    - ReLU: Applies the ReLU activation function to introduce non-linearity.
    - Another convolution layer follows to further process the feature maps.
  - pool1 (First Pooling Layer):
    MaxPool3d(2): A 3D max-pooling layer that reduces the spatial dimensions (height, width, depth) by a factor of 2.
  - enc2 (Second Encoding Block):
    Similar to the first block, this section uses two convolution layers (each followed by BatchNorm3d and ReLU) but increases the number of channels from 32 to 64.

- pool2 (Second Pooling Layer):

  A second max-pooling layer reduces the spatial dimensions further.

- Bottleneck: The deepest part of the network and contains the most abstracted representations of the input data.

  - bottleneck (Bottleneck Block):

    This block uses two convolution layers (with 128 filters) to process the data that has been downsampled through the encoder. Batch normalization and ReLU are applied after each convolution to maintain stable training and introduce non-linearity.

- Expanding Path (Decoder): This part of the network is responsible for increasing the spatial dimensions of the feature maps to reconstruct the segmentation output.

  - upconv2 (First Upsampling Layer):

    ConvTranspose3d(128, 64, kernel_size=2, stride=2): A transposed convolution (also known as a deconvolution) that upsamples the feature maps, doubling the spatial dimensions.

  - dec2 (Second Decoding Block):

    After upsampling, the feature maps are concatenated with the corresponding feature maps from the encoder (using skip connections), and then passed through a second decoding block that has two convolutional layers (64 filters).

  - upconv1 (Second Upsampling Layer):

    Similar to upconv2, this layer upsamples the feature maps from 64 channels to 32 channels.

  - dec1 (First Decoding Block):

    The final decoding block processes the concatenated feature maps and applies two convolution layers to refine the feature maps. The number of filters is reduced back to 32.

- Output Layer

  - final (Final Convolution):

    The final convolution layer reduces the number of channels to 1, which is suitable for binary segmentation (outputting a single channel, typically indicating the presence or absence of a certain class of objects). The kernel size is 1x1x1, meaning each voxel is classified independently.

Initially, we had not included batch normalization in the architecture, and it did not yield desirable results so we added batch normalization and consequently received improved results. Additionally, we tried adding dropout to further improve the performance of the model. This resulted in a much slower learning process which in turn did not improve the results. So the final architecture we kept does not include dropout.

To tackle the challenge of computational power, and to leverage the advantages of combining models, we trained a separate U-Net model for each particle type, and later combined the prediction from each of these models to produce the final predicted mask.

## 4.3 Ensemble of Specialized U-Nets

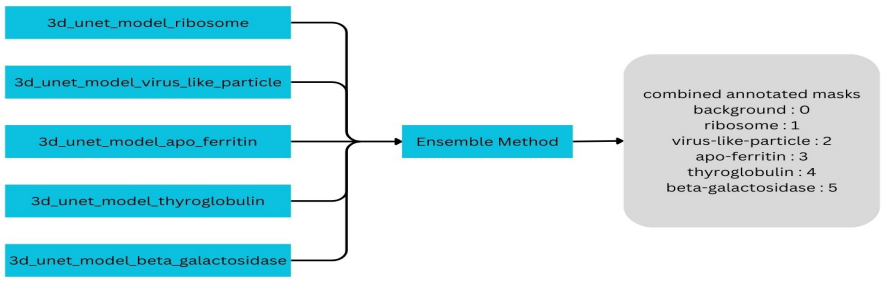


Figure 5: Pipeline of the 3D U-NET-based Model

After training separate U-Net models for each particle type, each model outputs a binary mask for its specific particle. To resolve conflicts between these different outputs, we combine them using an ensemble strategy that will be explained in the following section. The final output will be an annotated mask, where background voxels are labeled as 0, and the particles are labeled from 1 to 5 (figure 5).

## 4.4 Implementation

The configurations we used for training the U-Net models are:

- Loss Function: We chose Dice loss to improve segmentation accuracy. Dice loss directly optimizes overlap between predictions and ground truth.
- Optimizer: We used AdamW optimizer with a learning rate of 0.0001.
- Batch Size: After facing crashes due to computational costs, we reduced the batch size to 2 subvolumes per batch.
- Epochs: We trained the model for 400 epochs to track the improvement of the learning process and prevent the model from overfitting/underfitting, based on validation Dice score. Figure 6 shows the comparison of the training and validation losses for the apo_ferritin model. As you can witness, after around 50 epochs, the training loss shows minimal drops, while the validation loss does not show much improvement.
- Hardware: Training was performed on Google Colab GPUs.

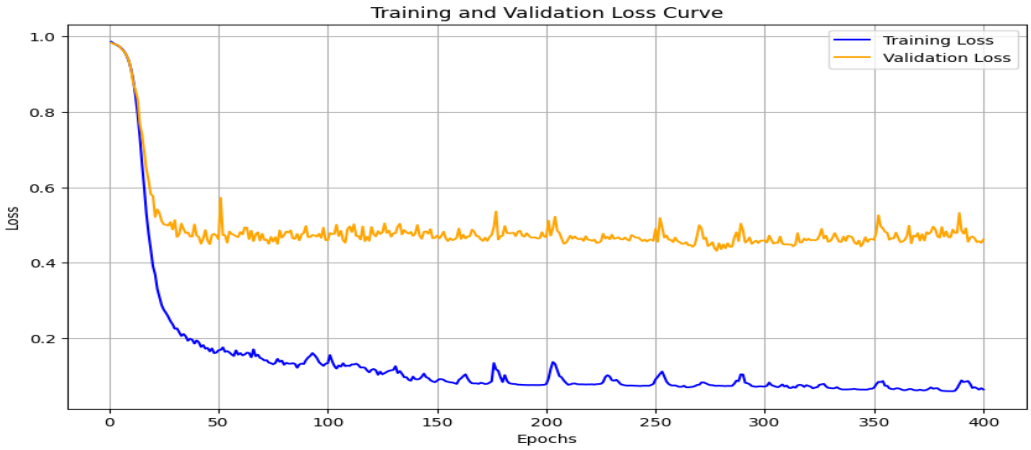


Figure 6: The decline of training and validation losses for 3d_unet_model_apo_ferritin

To create a unified annotated mask, we use an ensemble method:

- If all models predicted only background, then the final result should be all zeros
- Non-Overlapping Predictions: If only one particle is predicted for a voxel, it will directly be assigned to the combined mask using its particle ID.
- Overlapping Predictions: If more that one particle is predicted by different models for a voxel, we choose the class by weighing each class using the following criteria:
  - Particle Frequency: The more frequent particle type will be prioritized.
  - Model Recall: The model with higher recall for the overlapping particle is trusted more.
  - Probability Scores: Predicted probabilities (sigmoid outputs) were used as a tie-breaker, with higher confidence being favored.

  The class with the highest overall score will be chosen for the particle.

## 4.5 Results

The table below displays the results of evaluation of the different models. As mentioned by the outline of the kaggle competition, thyroglobulin and $\beta$-galactosidase were the most difficult particles to learn and predict, and out UNET model performed the poorest for these particles. On the other hand, the best model is the one trained to detect virus-like-particles and it resulted in a recall of 65.62%. Figure 7 shows the confusion matrix after evaluation of this model, and figure 8 compares the ground truth mask with the well-predicted mask by our model 3d_unet_model_virus_like_particle.

| Model | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 3d_unet_model_ribosome | 98.22% | 58.51% | 35.63% | 44.29% |
| 3d_unet_model_virus_like_particle | 99.41% | 80.61% | 65.62% | 72.35% |
| 3d_unet_model_apo_ferritin | 99.74% | 63.51% | 57.61% | 60.42% |
| 3d_unet_model_thyroglobulin | 99.07% | 50.10% | 33.74% | 40.32% |
| 3d_unet_model_beta_galactosidase limited | 99.45% | 55.41% | 26.13% | 35.51% |

Table 1: Evaluation of respective models for each particle

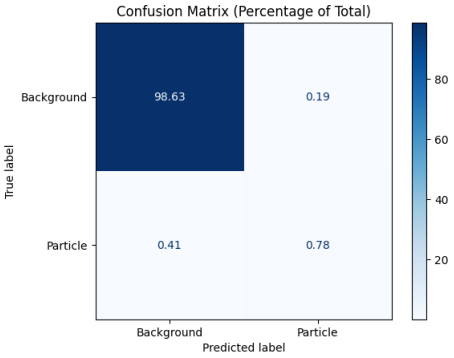


Figure 7: Confusion matrix for the virus-like-particles model

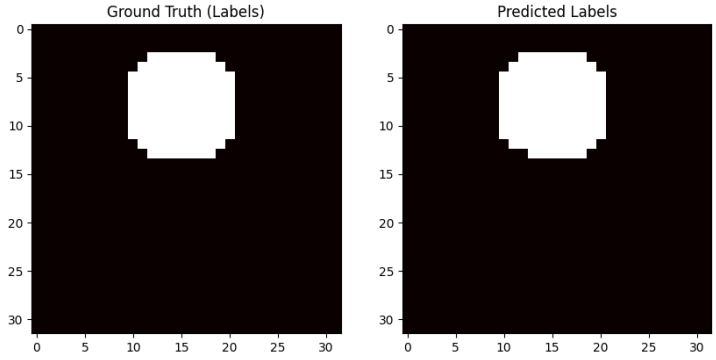


Figure 8: Comparison of the grand truth and the predicted virus-like-particle

# 5 Model 2 : VNET

## 5.1 Model Architecture

VNet is a deep-learning architecture designed specifically for volumetric medical image segmentation. Unlike traditional 2D networks, VNet operates on 3D data, making it ideal for tasks such as tumor detection, and other 3D imaging challenges. I was constructed as a 3D equivalent for the UNet architecture. The key difference between UNet and VNet apart from the dimensionality of the data they handle is that UNet uses skip connections while VNet Incorporates residual connections, where the input to a block is added to its output. Another difference is that while UNet uses deconvolutions for upsampling, VNet combines deconvolutions and residual pathways for upsampling.

The architecture of the VNet used during this project is as follows:

- Input Layer:
  The model takes an input tensor with a single channel.
- Encoder:
  The encoder contains residual blocks, each consisting of 3D convolutional layers, group normalization, and PReLU activations. Each residual block is followed by a strided convolution to reduce spatial dimensions by half.
  - Encoder 1: 1 input channel  8 output channels.
  - Encoder 2: 8 input channels  16 output channels.
  - Encoder 3: 16 input channels  32 output channels.
- Bottleneck:
  The bottleneck consists of a residual block that processes the feature map from 32 input channels to 64 output channels. It serves as the deepest layer in the network.
- Decoder:
  The decoder contains up-convolution layers for upsampling, followed by residual blocks. The upsampled feature maps are combined with corresponding encoder feature maps via skip connections (element-wise addition).
  - Up Convolution 3: Upsamples from 64 to 32 channels and combines with the feature map from Encoder 3.

- Decoder 3: Processes the combined feature map, reducing from 32 channels to 32 channels through a residual block.
- Up Convolution 2: Upsamples from 32 to 16 channels and combines with the feature map from Encoder 2.
- Decoder 2: Processes the combined feature map, reducing from 16 channels to 16 channels.
- Up Convolution 1: Upsamples from 16 to 8 channels and combines with the feature map from Encoder 1.
- Decoder 1: Processes the combined feature map, reducing from 8 channels to 8 channels.

- Output Layer:
  A final 1x1 convolution reduces the feature map from 8 channels to the number of output classes (e.g., 1 for binary segmentation). The raw output is returned without an activation function, allowing flexibility for loss functions.

## 5.2 Implementing Models

We initialize the 3D segmentation model, VNet3D, with an Adam optimizer (learning rate 0.00001) and a binary cross-entropy loss function (BCEWithLogitsLoss) configured with a pos_weight of 5.0 to address class imbalance. For each particle, the model is trained with the corresponding particle radius. We train the model for 100 epochs, where each epoch involves forward passes on training batches, loss computation, and parameter updation via backpropagation and the optimizer. The epochs average training loss is calculated and recorded. We validate the model in evaluation mode by calculating the average loss over the validation dataset without computing gradients and log the results every 10 epochs.

We calculate precision, recall, and F1 score to evaluate the model's performance on the training dataset. We collect and flatten the ground truth and predicted labels, threshold the predictions at 0.5, and compute the metrics to assess model accuracy and balance.

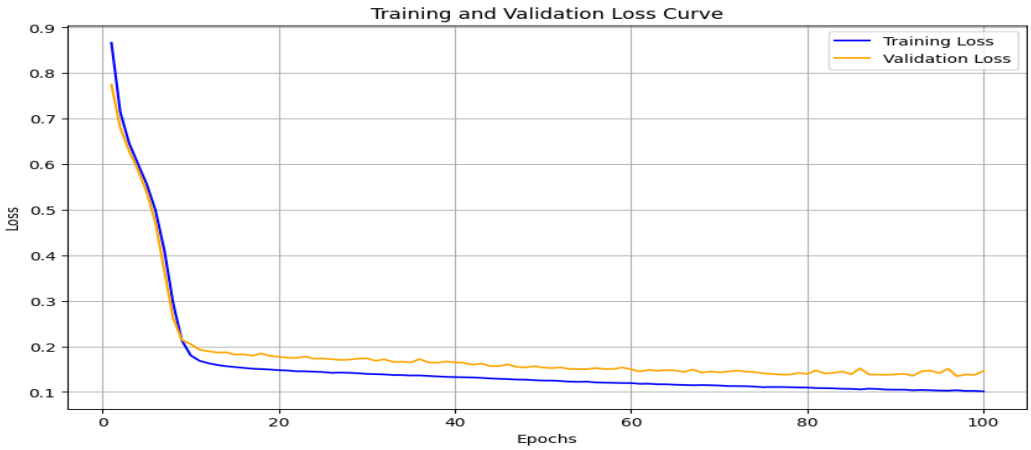


Figure 9: The decline of training and validation losses for VNet_model_ribosome

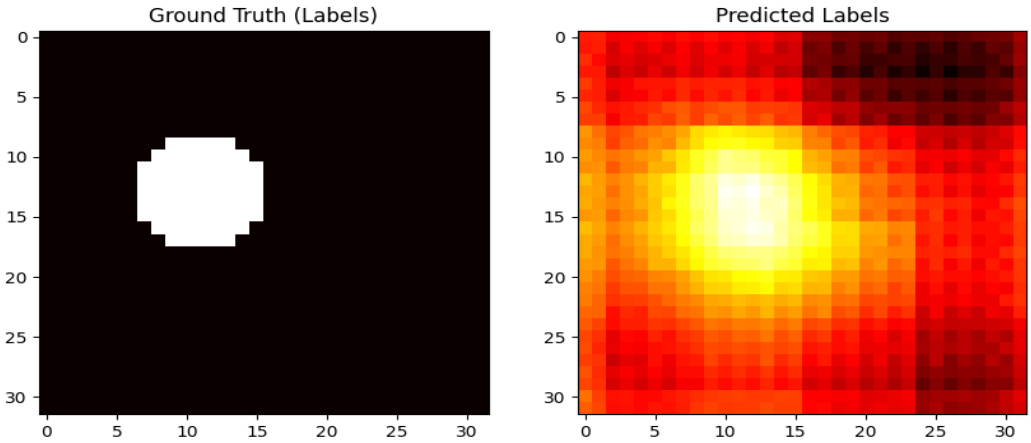


Figure 10: Comparison of ground truth and predicted mask for ribosome using VNet

### 5.3 Results

| Model | Precision | Recall | F1 |
|---|---|---|---|
| vnet_model_ribosome | 50.62% | 58.02% | 54.07% |
| vnet_model_virus_like_particle | 56.99% | 49.77% | 53.14% |
| vnet_model_apo_ferritin | 41.40% | 38.07% | 39.66% |
| vnet_model_thyroglobulin | 51.64% | 42.60% | 46.69% |
| vnet_model_beta_galactosidase limited | 30.28% | 30.51% | 30.39% |

Table 2: Evaluation of respective models for each particle

## 6 Future Work

The next step for this project is improving the results in both approaches by changing the comlexity of the models, creating an ensemble model with VNet models for each particle to obtain results on the whole dataset, and provide predictions in the format that is required by the kaggle competition to submit them on the kaggle website.

## 7 Comparison of the Architectures

U-Net is better overall, especially for virus-like particles and apo ferritin datasets. V-Net demonstrates strength in certain challenging datasets (ribosome, thyroglobulin, beta-galactosidase). For idetifying all the particles, creating an ensemble model with 3D U-Net models trained on virus-like-particles and apoferritin and VNet models trained on ribosome, thyroglobulin and beta-galactosidase will be an effective strategy to get good recall score for the entire dataset.

## 8 Conclusion

In this project, we explored the application of deep learning models, specifically 3D U-Net and VNet, for the segmentation and classification of protein complexes within cryo-electron tomography (cryoET) data. The results demonstrated the strengths and limitations of both architectures. The 3D U-Net showed superior performance for virus-like particles and apo-ferritin, while VNet

excelled in identifying ribosomes, thyroglobulin, and beta-galactosidase. These findings highlight the value of leveraging model-specific strengths through an ensemble approach for improved performance across all particle types.

Despite these advancements, challenges such as class imbalances, computational constraints, and the inherent complexity of cryoET data remain. Future work will focus on integrating ensemble methods with both architectures, and optimizing results for submission to the Kaggle competition. These efforts aim to contribute to the broader goal of automating protein identification in cryoET tomograms, a critical step in understanding cellular function and advancing human health.

# 9 Work Distribution

The work was distributed as follows:

- Fatemeh Gol Pour
  - Implemented simple 3d UNet to understand the problem
  - Implemented 3d UNet and interpreted the results
  - Wrote abstract, introduction, evaluation metrics, and model 3d UNet
  - Worked on formatting and editing the report
- Anna Karolin
  - Worked in visualizations to understand the data
  - Implemented VNet and interpreted the results
  - Wrote model VNet, comparison of architectures and conclusion
  - Worked on formatting and editing the report

# References

[1] Arnick Abdollahi, Biswajeet Pradhan, and Abdullah Alamri. Vnet: An end-to-end fully convolutional neural network for road extraction from high-resolution remote sensing data. *IEEE Access*, 8:179424 – 179436, 09 2020.

[2] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation, 2016.

[3] zgn iek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.