Protein Descriptor Neural Network Project - Documentation Summary

# 1. Overview

This document outlines the design and implementation of a neural network system for protein structure comparison using BioZernike descriptors. The system computes pairwise similarity between protein structures represented as a combination of geometric and Zernike features, then trains a classifier to predict structural similarity.

# 2. Dataset Design

The system supports two modes of operation:  
- Streaming Mode: Computes features on-the-fly from the protein dataframe using `StreamingProteinPairDatasetV2`.

- Cached Version [Dev mode branch only]

# 3. Descriptor Design

The descriptor is composed of two parts:  
- Geometric Feature Distance: Uses the formula `2 \* |a - b| / (1 + |a| + |b|)`.  
- Zernike Feature Distance: Uses `|a - b|` for each element.  
These are concatenated into a full 3922-dimensional vector.

# 4. Model Architecture

The model is a simple feedforward classifier with configurable hidden dimensions.  
It uses:

- n\_hidden\_dimension  
- ReLU activation  
- Final sigmoid output to predict similarity

# 5. Training Pipeline

The training loop is implemented in `train.py`, supporting both cached and streaming data loaders. Evaluation metrics are logged every epoch:  
- Binary Cross Entropy Loss  
- ROC AUC  
- PR AUC  
- Matthews Correlation Coefficient (MCC)  
  
TensorBoard support is integrated, with logs written to `tensorboard\_logs/`. Best models and epoch-wise checkpoints are saved in `modelData/`.

# 6. Evaluation and Baselines

Evaluation was conducted on the ECOD dataset. Final test metrics achieved:  
- ROC AUC: 0.965  
- PR AUC: 0.788  
- MCC: 0.704  
  
These results are comparable to the BioZernike NN baseline reported in the PLOS Computational Biology paper.

# 7. Key Design Choices

- Transitioned from nC2 generation to streaming/generator-based approach to reduce memory usage.  
- Implemented clean OOP-based `DescriptorInterface`, `GeometricFeature`, and `BioZernikeMoment` classes.  
- Supported both on-the-fly experimentation.  
- Enabled CPU-bound training with `torch.set\_num\_threads()` and optimized DataLoader usage.

- Used batchnorm, droput and relu for stable loss convergence

- Used BCEWithLogitsLoss with ReduceLROnPlateau optimizer as the optimal loss between 1e-5 and 1e-6

- Loggers, Tensorboard loggers enabled.  
- The code supports best datapoints and every epoch checkpointing