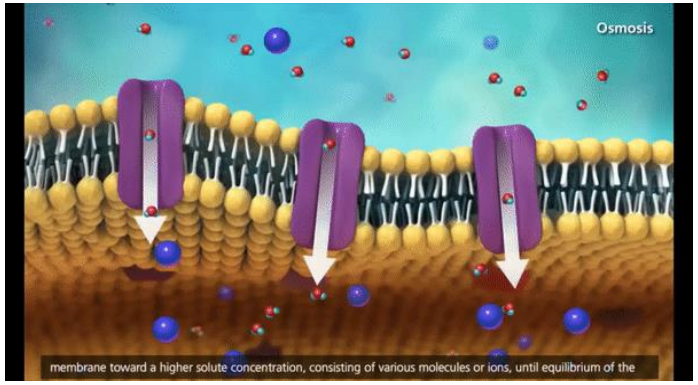
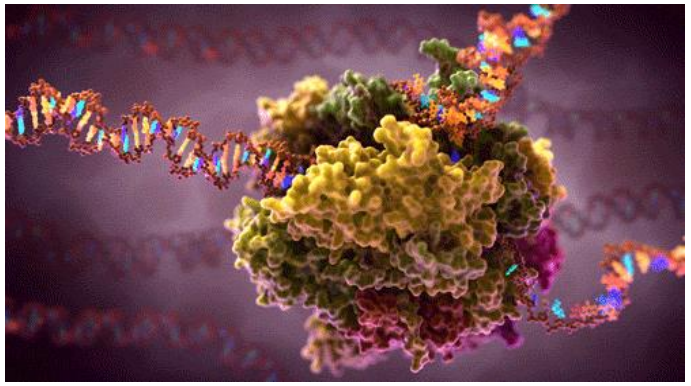
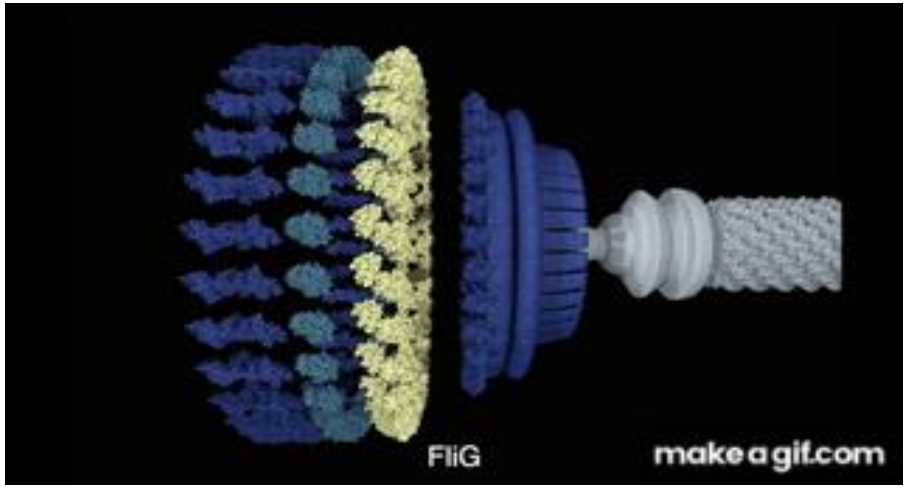
Deep Learning: Protein Structure

Experiment tracking Keras Neural Networks and Convolutional Neural Networks with mlflow for protein secondary structure classification.

Proteins are physiological workhorses!

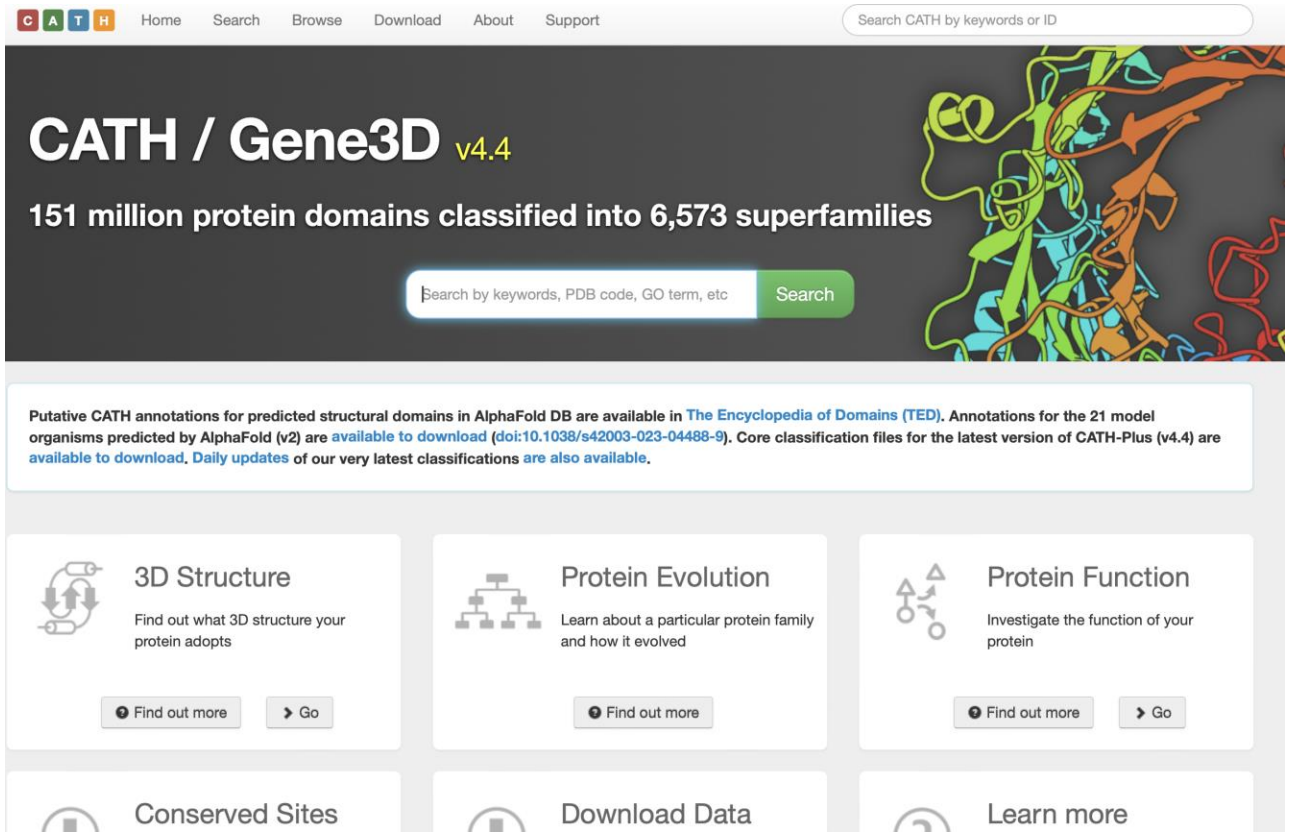






1. Catalysis biochemical reactions of metabolic pathways
2. Provide structure for cells / tissue
3. Responsible for transport
4. Liaisons for signaling and communication
5. Defense and Immunity
6. Cellular / organismal movement
7. Regulation of many processes (like gene expression or metabolism)
8. Storage and sensation
9. DNA Replication and Repair

CATH Database

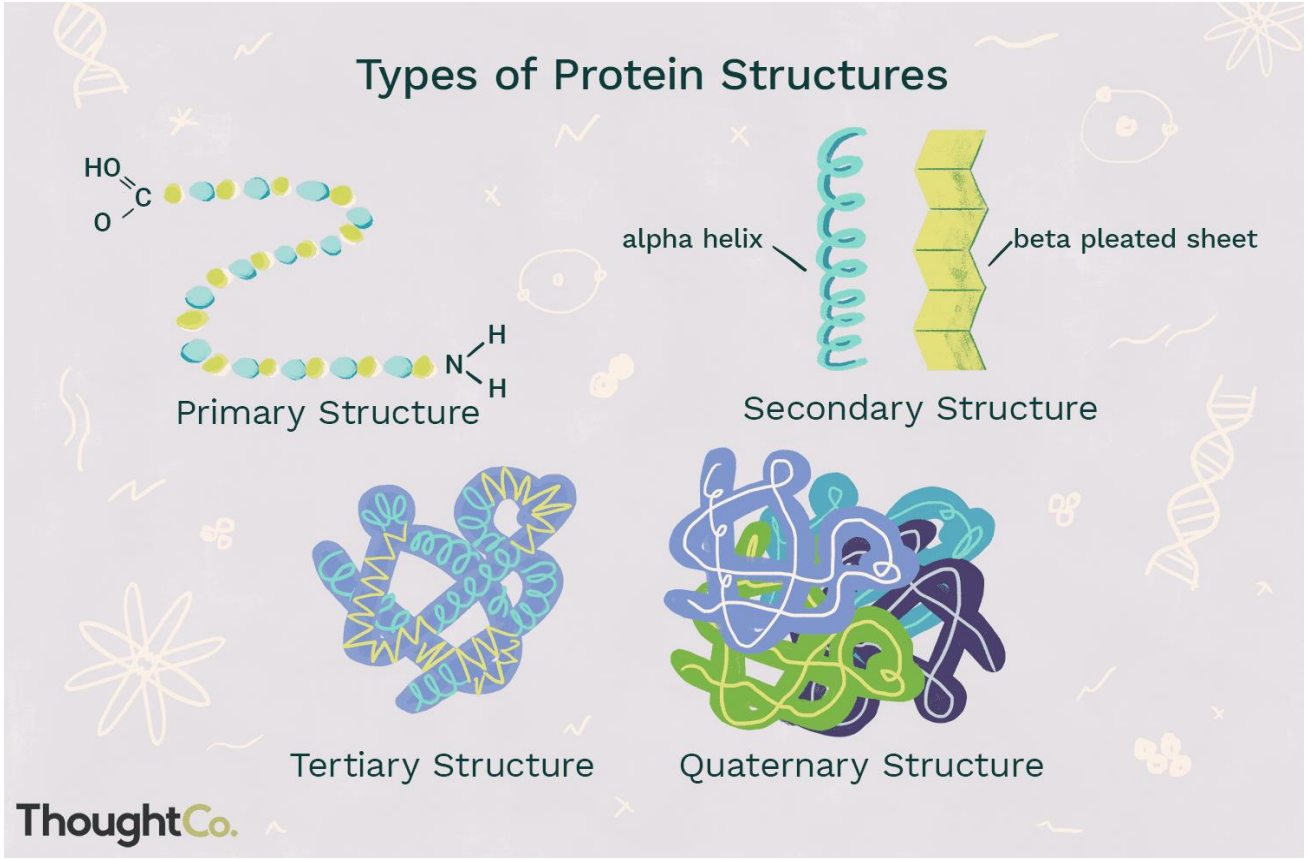


- The CATH database was created in the mid-1990s by Professor Christine Orengo and colleagues at University College London.
- It uses a combination of automatic methods and manual curation to classify protein domains from experimentally determined structures in the Protein Data Bank.
- The original goal of this classification was to help in identifying relationships between protein structures.
- In biology, it is known that structure informs function.
- **Understanding structure can lead to the understanding of protein evolution and function!**

Predicting Protein Structure

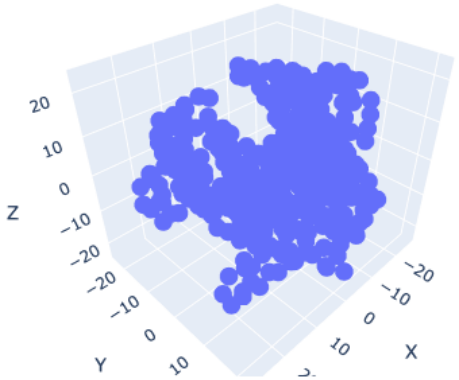
For this work, I am focusing on the `cath_3class.npz` dataset

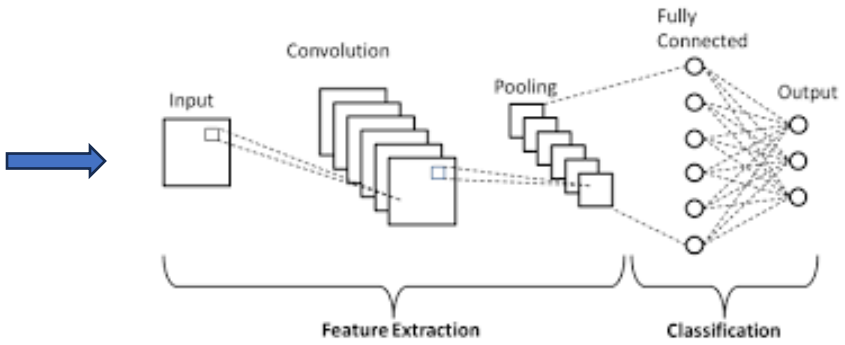
- This dataset focuses on the Class level of the CATH hierarchy.
- **Class (C):** Domains are categorized based on their secondary structure content. The main classes are:
  - Mainly Alpha (predominantly α-helices)
  - Mainly Beta (predominantly β-sheets)
  - Mixed Alpha-Beta (significant amount of both α-helices and β-sheets)
  - Few Secondary Structures



Deep Learning Approach

- Unlike classical machine learning problems that are based on making regressions or classifications on tabular data, the data used here is made up of 3D arrays of (x, y, z) coordinates.
- This is not easily solvable by classic ML methods line logistic regression, random forests, or even gradient boosted trees.
- Deep learning, and specifically, convolutional neural networks are a great application for this type of problem.





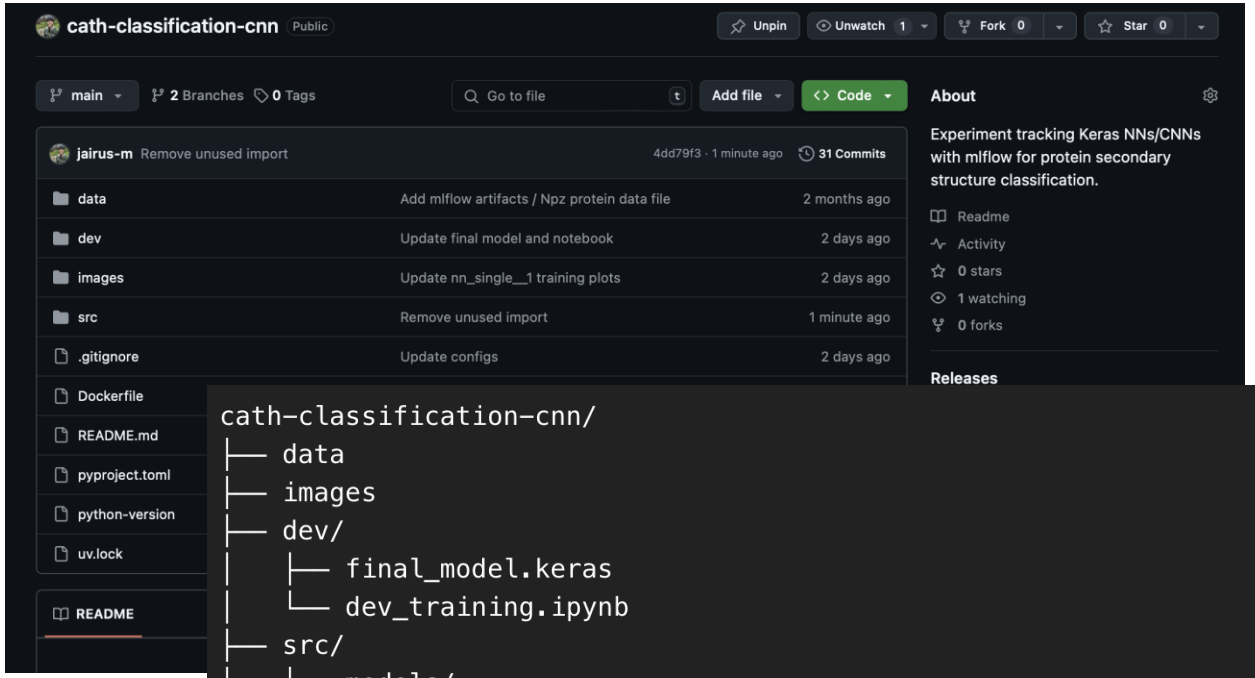
1. Mainly Alpha?
2. Mainly Beta?
3. Alpha-Beta?

Raw 3D Spatial Data

Convolutional Neural Network

Class Predictions

GitHub Repo




```
cath-classification-cnn/
├── data
├── images
├── dev/
│   ├── final_model.keras
│   └── dev_training.ipynb
├── src/
│   ├── models/
│   │   └── experimental_models.py
│   ├── utils/
│   │   ├── cath_data.py
│   │   └── process_data.py
│   └── main.py
├── Dockerfile
├── pyproject.toml
└── uv.lock
```

https://github.com/jairus-m/cath-classification-cnn

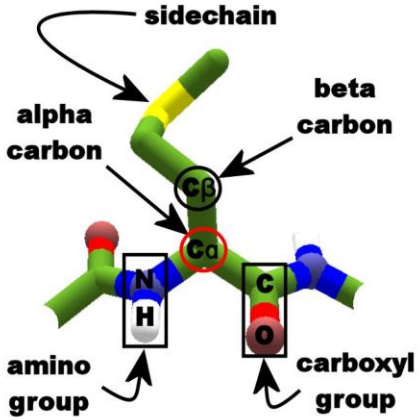
- Structured project
- Uses mlflow for experiment tracking
- Contains all the development EDA/training and production scripts to re-produce the experiments
- All dependencies in virtual environment is packaged up and can be ran in a Docker container

Inputs + Pre-processing

```
array([[-0.46251011, 14.0216713 , -0.11604881],
       [ 2.52248955, 14.92267227,  2.01995087],
       [ 0.57148933, 16.07967186,  5.23795128],
       ...,
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ]])
```

`protein_data[0]` aka protein #1

- 100% Class balance
- General Raw Input
  - Single Numpy Ndarray of 3D atomic positions
  - Shape (16962, 1202, 3)
    - (# of proteins, max # of AAs, 3D coordinates)
- Neural Network Pre-processing
  - Flatten to a single array of shape (16962, 3606)
    - Where 3606 == 1202 x 3
- CNN Pre-processing
  - Add spatial/channel dimensions and convert to tensors to yield shape:
    - (16962, 1, 3, 1)



Model Architecture

Model Architectures

1. **One-Layer NN**
   - 4 Neurons
2. **Two-Layer NN**
   - 64 Neurons
   - 64 Neurons
3. **CNN**
   - 2 3D Convolutional Layers
   - 2 Pooling Layers
   - 2 Dense Layers
4. **Simplified CNN w/ Early Stopping**
   - No Dense Layers

```
def cnn__4(X_train, X_val, X_test, y_train, y_val, y_test, signature):
    """
    Experiment 4: Simplified CNN with Early Stopping
    """
    with mlflow.start_run(run_name="Simplified CNN w/ Early Stopping"):
        model = keras.Sequential(
            [
                layers.Conv3D(
                    32,
                    kernel_size=(3, 1, 3),
                    activation="relu",
                    padding="same",
                    input_shape=(1202, 1, 3, 1),
                ),
                layers.MaxPooling3D(pool_size=(2, 1, 1), padding="same"),
                layers.Conv3D(
                    64, kernel_size=(3, 1, 3), activation="relu", padding="same"
                ),
                layers.MaxPooling3D(pool_size=(2, 1, 1), padding="same"),
                layers.Flatten(),
                layers.Dense(3, activation="softmax"),
            ]
        )
        model.compile(
            optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
        )
        early_stopping = EarlyStopping(
            monitor="val_loss", patience=2, restore_best_weights=True
        )
```

Model Results - mlflow

Run details

Run ID:	77eb0bf931d84c638825091555cc6991	b8444e199a5a408ebdbeed48798dcb0f	05d2f45ce16c475e93f9cfab6f813844	b054d14d51b84041ac491d1adf22b883
Run Name:	Simplified CNN w/ Early Stopping	CNN	Basic NN - Two Layers	Basic NN - One Layer
Start Time:	2025-02-19 20:45:33	2025-02-19 20:24:02	2025-02-19 20:23:52	2025-02-19 20:23:46
End Time:	2025-02-19 20:53:09	2025-02-19 20:45:33	2025-02-19 20:24:02	2025-02-19 20:23:52
Duration:	7.6min	21.5min	10.1s	6.0s

Parameters

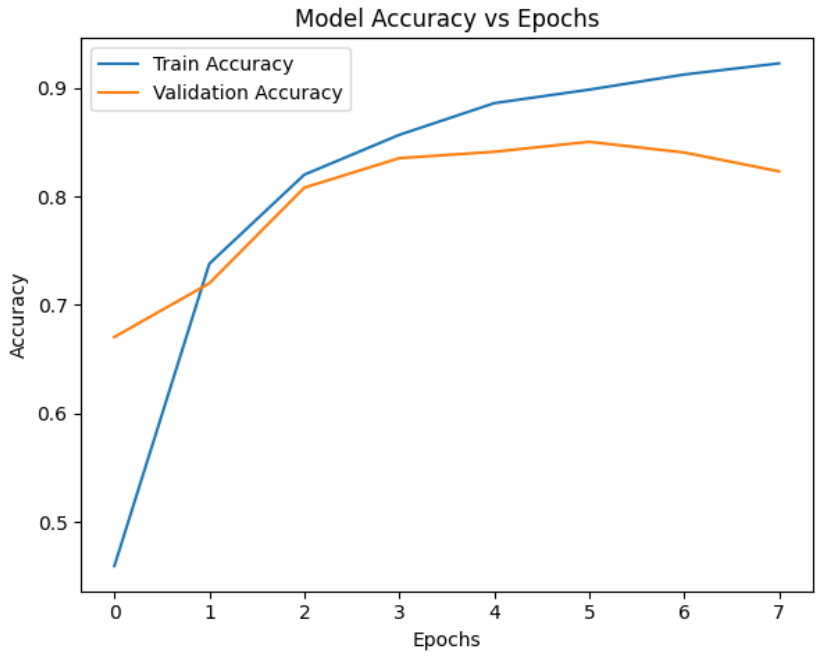
Metrics

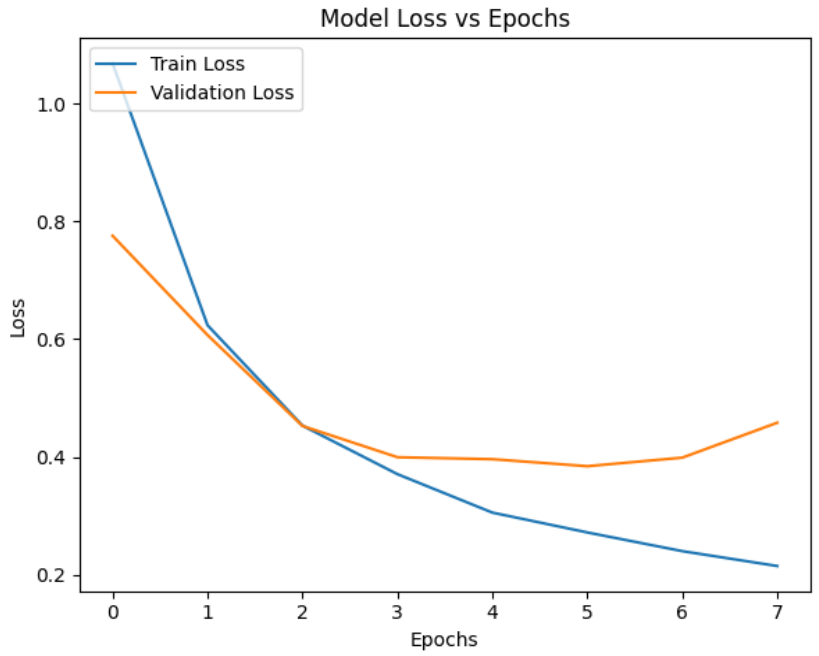
Show diff only

test_accuracy	0.856	0.852	0.615	0.37
test_loss	0.386	0.832	1.679	1.185

Final Results

- **All experimental results**
- **Final model:** Simplified CNN w/ Early Stopping
  - CNN Architecture
- **Training time:** ~7 min
- **Accuracy:** ~86%





Final Results – Output

- Training set raw predictions / results:

Raw Predictions

```
array([[9.6096075e-01, 6.6123974e-08, 3.9039128e-02],
       [3.1711245e-01, 4.5463727e-03, 6.7834109e-01],
       [1.2312427e-03, 6.8753022e-01, 3.1123856e-01],
       ...,
       [6.1975126e-03, 7.1378388e-02, 9.2242420e-01],
       [2.0903088e-03, 6.9562298e-01, 3.0228665e-01],
       [8.7979293e-05, 5.9445584e-01, 4.0545610e-01]], dtype=float32)
```

Confusion Matrix/ Accuracy

```
Confusion Matrix:
[[555   0  15]
 [  4 480  80]
 [ 44  79 440]]
```

```
87
54/54 ━━━━━━━━━━━━━━━━━━━━ 1
Test accuracy: 0.8556
Test loss: 0.3735
```

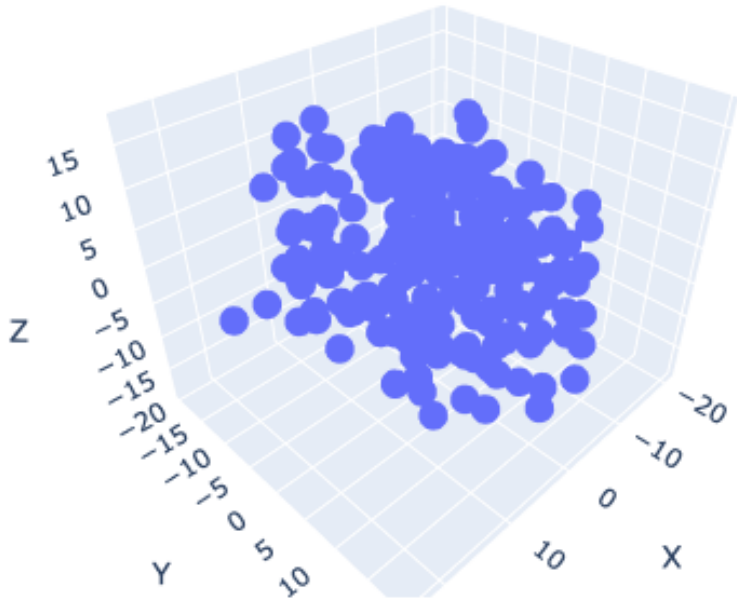
String Results

	y_true	y_pred	is_equal
0	Mainly Alpha	Mainly Alpha	True
1	Mainly Alpha	Mainly Alpha	True
2	Mainly Beta	Alpha-Beta	False
3	Mainly Alpha	Mainly Alpha	True
4	Mainly Beta	Mainly Beta	True
...	...	...	...
1692	Alpha-Beta	Alpha-Beta	True
1693	Mainly Beta	Mainly Beta	True
1694	Mainly Alpha	Mainly Alpha	True
1695	Mainly Alpha	Mainly Alpha	True
1696	Mainly Alpha	Mainly Alpha	True

1697 rows × 3 columns

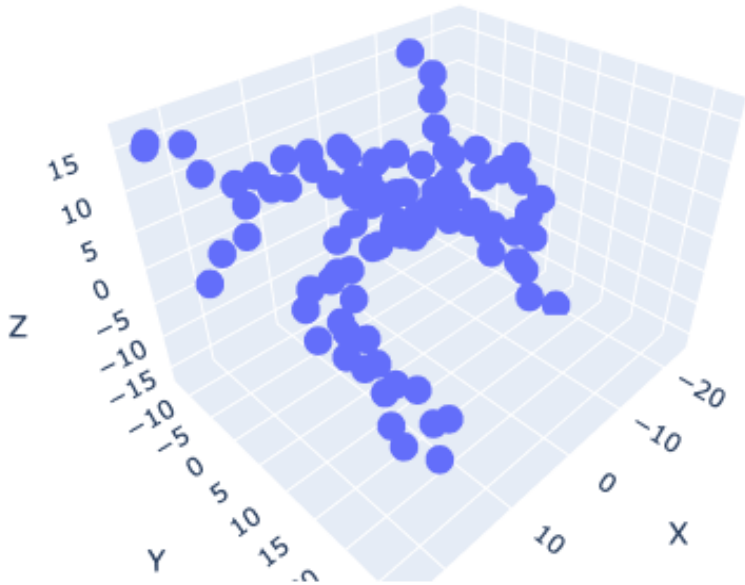
Example Results

Predicted 3D Structure of Protein #877: 'Alpha-Beta' (True Class: 'Alpha-Beta)



Example Results

Predicted 3D Structure of Protein #2: 'Alpha-Beta' (True Class: 'Mainly Beta)



Conclusion

- Iterated and developed different NNs and CNNs to solve a complex problem
  - Improve performance from 37% > 86%
- Built a Python project that utilized mlflow to organize and track experiments
  - Modular, version controlled, reproducible, and self-contained (Dockerized)
- Successfully used 3D atomic spatial data to predict protein secondary structures with reasonable accuracy