Protein Data Analysis

Group 7: Arjun Dharma, Thomas Waldschmidt, Rahil Dedhia

Advisor: Dr. Peter Rose

Director of the Structural Bioinformatics Laboratory at SDSC

June 5, 2020

Abstract

Deep Learning transformer models such as Bidirectional Encoder Representations from Transformers (BERT) have been widely successful in a variety of natural language based tasks. Recently, BERT has been applied to protein sequences and has shown some success in protein prediction tasks relevant to biologists, such as secondary structure, fluorescence, and stability. To continue the investigation into BERT, we examined a new prediction task known as subcellular location, first described in DeepLoc (2017). Using BERT embeddings from a UC Berkeley research project titled Tasks Assessing Protein Embeddings (TAPE) as features for downstream modeling, we achieved a 67% test set accuracy using a support vector classifier for the 10 class classification task, and 89% using a Keras deep neural network for the binary classification task (membrane bound vs water soluble protein). Next, we created a containerized Flask app using Docker which is deployable to AWS EC2 with the ability to run on a GPU. This service allows for embedding protein sequences using pretrained models, as well as providing an interface for visualizing the embedding space using principal component analysis and plotly.

Contents

1 Introduction

1.1 Challenge and Problem Statement

1.1.1 Understanding Proteins

Proteins are a fundamental building block of life. They perform complex functions ranging from transporting oxygen in the body, detecting stimuli, providing structure to cells, and even DNA replication. Proteins consist of a linear chain of 20 possible amino acids connected by covalent bonds. This sequence of amino acids can be represented as a sequence of discrete tokens, known as the primary structure. The unique combination of these amino acids give rise to unique functions which are determined by the protein's three-dimensional structure. The local geometry of a protein is known as secondary structure and describes the behavior of small segments of proteins, either taking the form of an α -helix or a β -sheet. The three dimensional geometry of the protein is known as the tertiary structure, and describes the overall function of the protein. Figure 1 shows a visual representation of the various levels of protein structure.

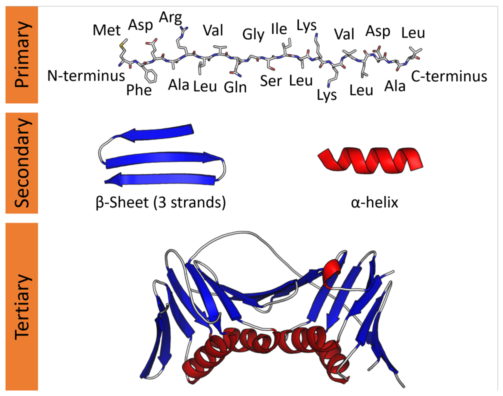


Figure 1: Summary of protein structure (primary, secondary, and tertiary) [1]

There has been a large amount of work in sequencing proteins and storing them within large scale data repositories, such as UniProt [16], which contains hundreds of millions of protein sequences, a majority of them unlabeled. Currently, there are about 300,000,000 sequences in UniParc, and about 160,000 structures in the Protein Data Bank (PDB) [4]. Determining the structure and functions of proteins are both difficult and costly, where identifying a single structure takes upwards of \$100k [15]. The number of sequence proteins has significantly outpaced the capacity and resources for those sequences whose structures have been identified, shown in figure 2.

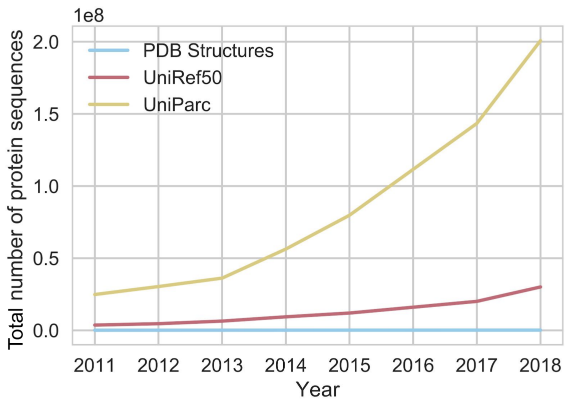


Figure 2: The relationship between the total number of sequences available versus the number of sequences that have been examined for structure [4]

A fundamental challenge of biochemistry is the prediction of protein structure and function given an input primary sequence. Existing research methods include comparing protein sequence families, or evolutionary similar proteins responsible for a unique function and require a large amount of sequences that must be labeled to be evolutionary similar. This method is not generalizable and isn't applicable to novel proteins. Additionally, this analysis is limited to proteins with a large number of family members. A significant amount of research has been done recently in applying AI and deep learning to study protein sequences and achieve success in a variety of protein prediction tasks. Instead of using a labeled subset of protein sequences, these methods aim to leverage large corpuses from UniProt in order to derive meaningful insight. One such endeavor is AlphaFold [2] by Google's DeepMind, where they built a system to predict protein structure and generate 3D images using large genomics datasets. Similarly, researchers have been applying state of the art natural language processing techniques to primary protein sequences in order to gain insights into common protein prediction tasks such as secondary structure, contact prediction, stability, and fluorescence. One NLP model that has received recent attention is the transformer architecture BERT (Bidirectional Encoder Representations from Transformers), which has demonstrated success in forming deep understanding of unlabeled texts through pretraining.

1.1.2 Transformer Architecture and BERT

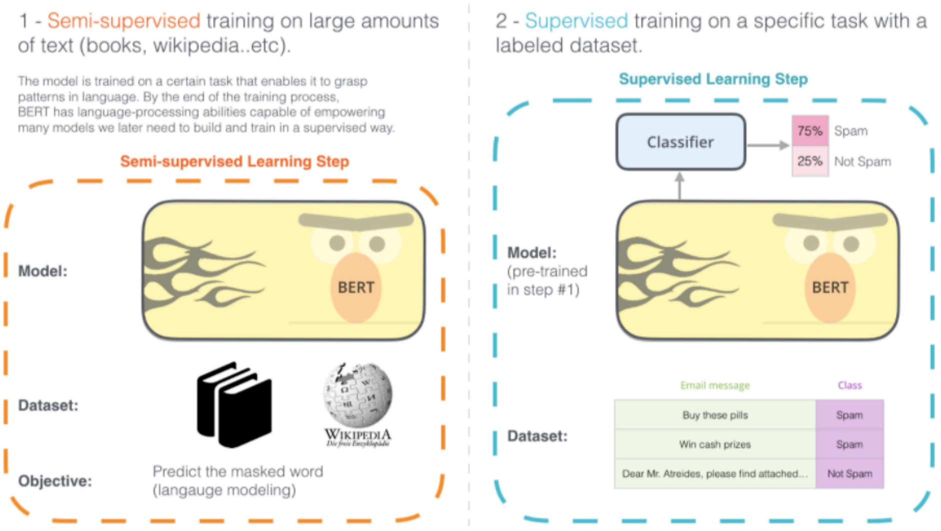
Natural language processing (NLP) is a field concerned with how computers understand and process large amounts of data in every day language. Some challenges in NLP include speech recognition, language generation, translation, and language understanding. Traditional methods were based on complex sets of hand-written rules; however, since the 1980s many have turned to machine learning as an approach to language processing. In the early 2010s, deep learning style machine learning methods began to be applied to traditional natural language processing techniques due to an increase in data and compute power. [3]

Many techniques have been developed in order to represent natural language in a machine readable form. Embeddings, a vectorized representation of language in the form of words, sentences, or documents are useful tools in turning sequences of text into features. Embeddings capture statistical relationships between words or phrases in a body in text to form a high dimensional feature space which can then be input as a numerical representation for NLP tasks. Embeddings can be generated through creating a co-occurrence matrix, or neural networks in the form of models like word2vec, ELMo, and BERT. Neural networks trained on large corpuses of texts create language models which have been widely successful in well known NLP benchmarks described in the General Language Understanding Evaluation (GLUE), a collection of resources for training, evaluating, and analyzing natural language understanding systems [5]. These tasks include question answering, semantic similarity, paraphrasing, and sentiment. Recently, these tasks have been dominated by variations of BERT, a type of transformer model.

First introduced in 2017, the Transformer is a deep learning model which has achieved wide spread success in NLP. The Transformer architecture consists of an set of encoders and decoders, each with their own attention mechanism and feed-forward neural network. Attention allows a model to understand a sentence as a whole and provides information on tokens farther away from a given word. Transformers leverage the power of this attention mechanism to provide large gains in both compute performance and downstream task performance. These models can be trained on large bodies of unstructured text to gain a deep understanding on a variety of tasks.

BERT, or Bidirectional Encoder Representations from Transformers is a method of pretraining language models on large corpuses of text such as Wikipedia for downstream NLP tasks such like question answering or summarization. BERT is both unsupervised and bidrectional in its pre-training.

While context-free models such as word2vec or GloVe generate an embedding for a given word, BERT generates a representation of a word or sequence based on the words in either beginning or end of a sentence. BERT also leverages a semi-supervised form of pretraining, where some tokens are masked in the input when passed through the bidirectional Transformer encoder and are asked to predict the masked words [6]. Additionally, some BERT models use next sentence prediction as a way to understand relationships between input sequences. Using BERT consists of two stages, pretraining and fine-tuning, shown in Figure 3. The pretraining step is computationally intensive and requires large amounts of data, but the model can then be used as an encoding layer for later fine tuning. Fine tuning is a supervised task where BERT is adapted to the task at hand. The base pretrained model provides knowledge and insight from the large corpuses it is trained on and can be used as a base embedding layer for the fine tuning.


1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).
Semi-supervised Learning Step
Model:
BERT
Dataset:
WIKIPEDIA
Objective:
Predict the masked word (langauge modeling)
2 - Supervised training on a specific task with a labeled dataset.
Supervised Learning Step
Classifier
75% Spam
25% Not Spam
Model: (pre-trained in step #1)
BERT
Dataset:

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached...	Not Spam

Figure 3: BERT's pretraining and fine-tuning steps. Pretrained models can be exported to quickly do fine tuning for unique tasks. [7]

BERT and the Transformer's ability for self supervised pretraining in natural language has drawn attention to the immediate analogy for protein sequences. With the recent availability of large amounts of protein sequences in databases such as Unirep, these unlabeled sequences can be used to train language models like BERT to learn features that transfer to downstream tasks relevant in biochemistry and evolutionary biology. If trained on a sufficiently large corpus, can BERT learn the underlying grammar or linguistic structure of protein sequences, formed over generations of biology and evolution? Can this knowledge be used to better predict the structure and function of these proteins?

1.1.3 Protein Embeddings and TAPE

A primary sequence for a protein is represented as a sequence of tokens, such as

$$GCTVEDRCLIGMGAILLNGCVIGSGSLVAAGALITQ$$

Each amino acid can be one of 20 different types represented by the set [ARNDBCEQZGHILKMFPSTWYV]. These sequences can be "tokenized" by breaking up the sequence into its individual amino acids, similar to how sentences are tokenized into individual words or phrases. Traditionally, these sequences have been encoded using methods such as one-hot encoding, where a sequence is converted into a matrix that shows which amino acids exist at a given position by assigning it a value of 0 or 1. Paired with sequence alignment, a method of comparing sequences from evolutionary relationships, this method of encoding has achieved decent success when fed through neural network systems. Other methods of encoding evolutionary features can be found in techniques such as Blosum62, a method to score alignments between evolutionary divergent protein sequences [17]. Word2vec, another method mentioned before, has also provided a context-free representation of protein sequences on an individual amino acid level. Sequence specific encodings of amino acids have ranged from techniques like position specific scoring matricies (PSSM) and profile hidden Markov models (pHMM). Embedding representations generated by neural networks in the form of LSTMs, or residual neural networks have also been successful within downstream tasks.

Variations of BERT and the transformer architecture have been applied to protein sequences with some success [8] [9]. Different models leverage different datasets and are tested in different downstream tasks, making it slightly difficult to compare the quality of these embeddings. Similar to GLUE, mentioned in the previous section, Tasks Assessing Protein Embeddings (TAPE) is a collection of resources created by researchers at UC Berkeley that provide corpuses for pretraining, supervised downstream tasks, as well as benchmarking code. The five downstream tasks TAPE provides are secondary structure, contact map prediction, remote homology detection, fluorescence, and stability. The pretrained models they provide are a BERT base model, a deep representation model called UniRep, another deep learning model known as trRosetta [10], as well as infrastructure to train a ResNet, LSTM, and a one-hot encoded model.

Since pretraining a BERT model from a given corpus is computationally expensive, we leveraged the pretrained BERT model from TAPE to embed our data and conduct further analysis. Based on initial results, the BERT model achieved good results when benchmarked against other models on tasks such as secondary structure prediction, fluorescence, and stability. This shows that pretraining a BERT model on a large training corpus of unlabeled sequences provides base knowledge for a variety of tasks.

1.1.4 Subcellular Location

To further the exploration of their pretrained BERT model, we chose to focus on a new task not mentioned in TAPE, known as subcellular location. Subcellular location is a well-studied topic in bioinformatics research. Subcellular localization prediction involves predicting where a protein specifically resides in a cell, such as the nucleus, endoplasmic reticulum, or other organelles, shown in Figure 4. Predictions of these locations is an impor-

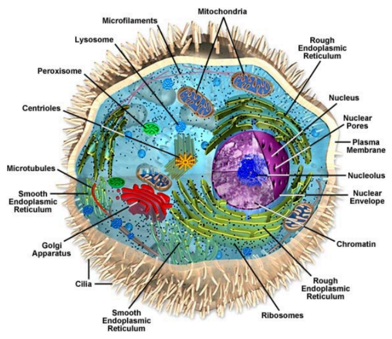


Figure 4: Diagram of subcellular location types[13]

tant component of bioinformatics to predict protein function and genome annotation, as well as identification of drug targets. Many machine learning techniques have been applied to this task, but most of them rely on annotations of similar proteins in a knowledge database.

The DeepLoc [14] dataset consists of labeled protein sequences for two subcellular prediction tasks. The first task consists of a ten class classification problem where a sequence is categorized as belonging to either the nucleus, cytoplasm, mitochrondrion, extracellular, cell membrane, endoplasmic reticulum, plastic, golgi apparatus, lysosome/vacuole, or peroxisome.

The second task consists of a binary classification task to denote if the protein is soluble or membrane bound, shown in Figure 5. In the original paper,

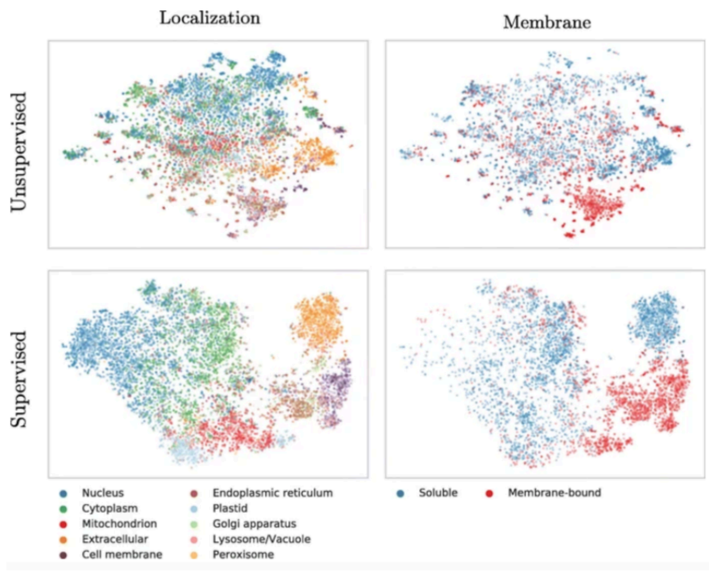


Figure 5: Example embedding space of DeepLoc using t-SNE [13]

the researchers leverage deep neural networks using an RNN as well as an attention mechanism trained on raw labeled protein sequences from UniProt. After hyperparameter optimization and adding information on protein profiles, they achieved a 78% accuracy on the 10 class task, a 92% accuracy on the binary classification task.

In this project, we aimed to benchmark BERT against existing models tested on DeepLoc. Using both deep learning methods as well as traditional machine learning methods, we use the TAPE pretrained BERT model for our analysis on subcellular location. Based on BERT's success in both NLP as well as protein prediction tasks, we hypothesized that BERT will also show success in subcellular localization.

2 Team Roles and Responsibilities

2.1 Assigned Roles

- Arjun Dharma - Record Keeper
- Rahil Dedhia - Project Manager

- Thomas Waldschmidt - Treasurer and Project Coordinator

2.2 Major Contributions

- Arjun Dharma- Sequence embedding, PCA visualizations, applying transformer model to subcellular location, flask app for embedding and visualization.
- Rahil Dedhia - Applying transformer model to subcellular location, multitask learning, Dockerized flask app for EC2 deployment
- Thomas Waldschmidt - Applying sklearn and Keras models to TAPE embeddings, model analysis with an emphasis on ROC curves

3 Methodology

We took two approaches in benchmarking our DeepLoc models against the state of the art. First, we leveraged the training capabilities of TAPE in order to add the DeepLoc dataset as a task in order to run against their BERT model, as well as to benchmark it against other models they have available. This system leverages the deep learning framework pytorch and involves training over GPUs. Next, we used the embedding capabilities of TAPE to embed the DeepLoc dataset to use to train less computationally intensive machine learning models like logistic regression and XGBoost. We discuss the findings of both the deep learning model for the 10 class classification task as well as the results from various models for the 10 class and binary classification tasks.

4 Data Processing

The DeepLoc dataset is composed of a total of 14404 sequences in fasta format, a format common to biological researchers. The file itself is only 7.7 MB and can simply be stored on disk. The original fasta file is transformed into formats useful for our later models. A diagram depicting the Data flow is shown in figure 7.

Each sequence has a unique protein ID, a label denoting if it is in the train or test set, and labels for the 10 class and binary classification tasks. The raw data set was passed through a notebook that parsed out these labels and generated individual fasta files for the train and test sets. The class distributions for the train and test sets are shown in figure 7.

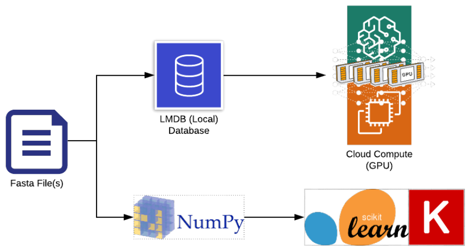


Figure 6: Data Flow from the raw fasta files into lmdb files and npz (numpy array) files for downstream modeling.

Nucleus	3235	Nucleus	808
Cytoplasm	2033	Cytoplasm	508
Extracellular	1579	Extracellular	393
Mitochondrion	1208	Mitochondrion	302
Cell.membrane	1067	Cell.membrane	273
Endoplasmic.reticulum	689	Endoplasmic.reticulum	173
Plastid	605	Plastid	152
Golgi.apparatus	286	Golgi.apparatus	70
Lysosome/Vacuole	257	Lysosome/Vacuole	64
Peroxisome	124	Peroxisome	30

Figure 7: Class distributions of the train(left) and test(right) set after filtering.

During the parsing step, we removed an ambiguous class for Cytoplasm/Nucleus, reducing our number of sequences to 13858. Next, we filtered out sequences larger than a max sequence length depending on our downstream analysis. For the 10 class deep learning model, we filtered out sequences larger than 1024 amino acids due to memory constraints on the GPU, resulting in 12704 sequences in our dataset. For the other models, we filtered out sequences larger than 6000 amino acids, removing only 2 of the sequences resulting in 13856 entries. After splitting the train and test sets, we generated a validation set from 10% of the train set while preserving class balances.

For the binary classification task, there were three possible classes in the dataset, S (soluble), M (membrane bound) and U (unknown). We removed entries corresponding to U to reduce the classification to purely binary in one model, but also explored using a masked loss function in a multiclass classification model that ignored those ambiguous classes. The resulting dataset for the binary task before embedding was 8,660. This reduced set was used for the Keras deep neural network as well as the multitask classifier described previously.

The TAPE deep learning training infrastructure required the fasta files to be in lmdb format, a data structure that provides high performance reads in the form of a key-value store. This required further processing to generate these lmdb formats, which then could be ingested into TAPE.

Next, the original dataset and the train/test/validation files were embedded with pretrained BERT features using the BERT model provided by TAPE. The embedded files are output as numpy array files which can be used for modeling using sklearn, xgboost, and keras. A given input sequence is embedded into a 768 dimensional vector of pretrained features generated from BERT. The embedding step also allowed us to explore the embedding space generated by BERT using principal component analysis (PCA) and visualization.

5 Analysis and Modeling

5.1 PCA and Visualization

Using principal component analysis, we were able to embed the BERT pretrained features by projecting the 768 dimensional data down to 3 dimensions. Principal component analysis is done by projecting the data onto its basis vectors, known as principal components. In these dimensions, the data is distributed to maximize variance. By embedding the full DeepLoc data set, we were able to visually verify the ability of BERT to represent these sequences in the feature space. The explained variance as a function of the number of components preserved is shown in figure 8.

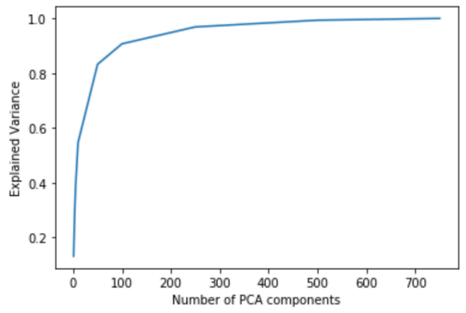


Figure 8: Explained Variance as a function of PCA components for DeepLoc

In two dimensions, only 20% of the variance is explained although the data shows clustering and segmentation. We see that around 150 dimensions, the explained variance is above 90 percent. The two dimensional PCA projection of the 10 class classification task is shown in figure 9

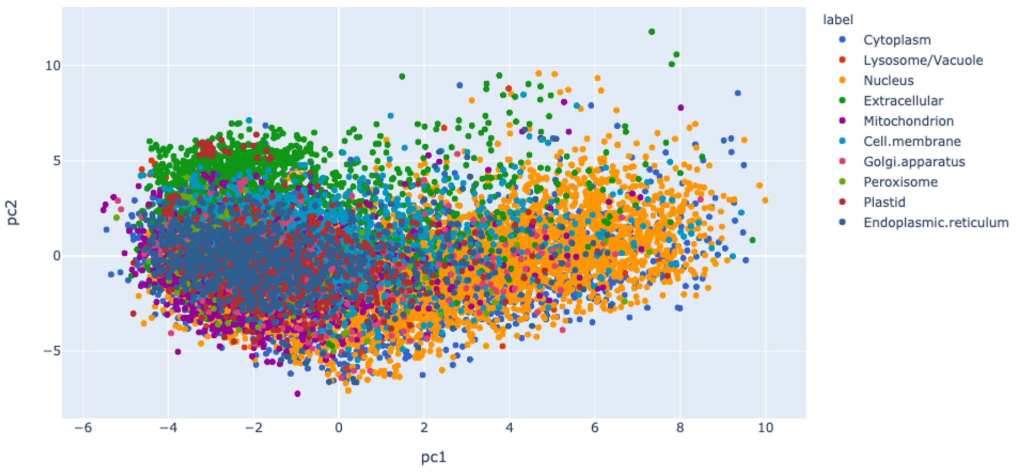


Figure 9: PCA in 2 dimensions of DeepLoc for the 10 class classification task.

For the 10 class task, the PCA projection contained 28% of the explained variance in 3 dimensions. However, the resulting visualization proved to be very insightful in depicting how well segmented and clustered the labeled sequences were in the resulting feature space. Two angles of the 3D visualization are shown below in figure 10. For the binary classification task, the embedding space is visualized in 2d in figure 11.

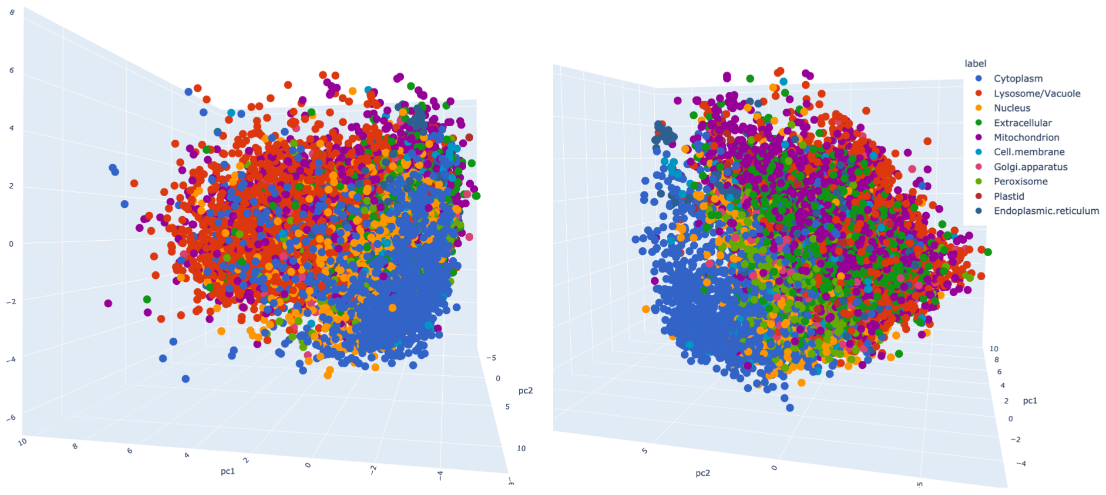


Figure 10: PCA in 3 dimensions of DeepLoc for the 10 class classification task.

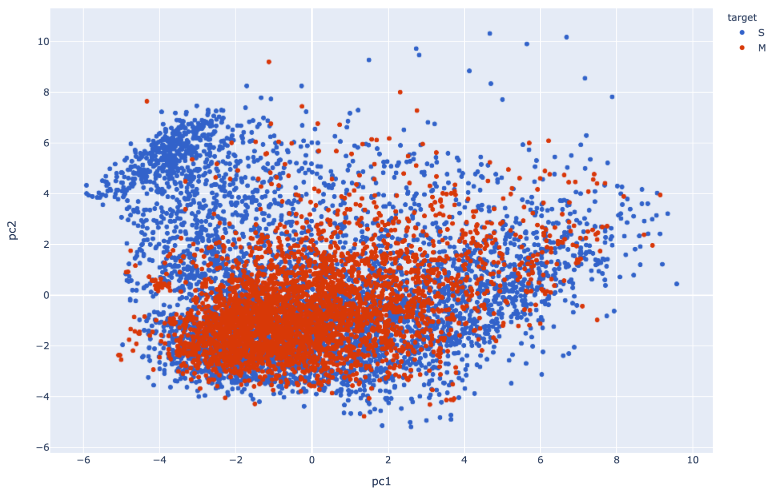


Figure 11: PCA in 2d of DeepLoc for the binary classification task

The PCA visualizations using BERT features outline's the model's ability to understand novel tasks that it has not been exposed to. Additionally, visually seeing how well the data was segmented in both 2 and 3 dimensions provided us with confidence that simple modeling approaches other than deep learning classifiers would work in this context.

5.2 DeepLoc Modeling

The full results of all the models used are shown in the table below. A more detailed description of the models and their parameters are explained in the following sections.

Model	Q10 Test Accuracy	Q2 Test Accuracy
Logistic Regression	66.2%	87.2 %
Support Vector Classifier	67.5%	
XGBoost	63.8%	
Keras Deep Neural Network	64.5 %	89.6 %
Multitask Deep Neural Network	66.2 %	72.6 %
Transformer(BERT)	61%	

5.2.1 Fine tuning BERT using Deep Learning and TAPE

After adding DeepLoc as a task to TAPE, we were able to benchmark the BERT transformer model against other models available in TAPE. We only evaluated the 10 class classification task for this step. The transformer model was trained on a p3.8xlarge EC2 instance on 4 GPUs. We used a batch size of 64, a learning rate of 7e-5, 1000 warm up steps, 20 epochs, and 4 gradient accumulation steps. For the other deep learning models, not enough sufficient hyper parameter optimization was made in order to maximize test accuracy. Further investigation into these models using TAPE will be useful in later research.

5.2.2 Sklearn, XGBoost, and Keras Modeling

Using the embedded train, test, and validation files, we generated numpy arrays to load into various downstream models. The XGBoost model's 63.1% accuracy was achieved with a 0.25 eta, a max depth of 10, a softprob objective for multi class. Our most successful model was a support vector classifier for Q10 in sklearn. This model was optimized using the GridSearchCV function in sklearn, resulting in a C value of 500, 100000 max iterations, and a polynomial kernel function.

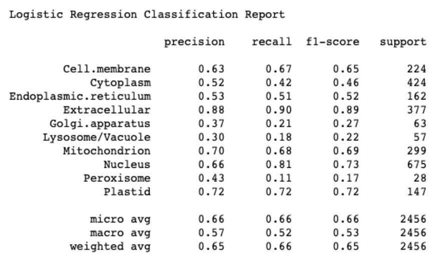
For the binary classification task, our most successful model was a Keras Deep Neural network. The Keras DNN for the 10 class task consisted of 2 hidden layers of 32 nodes with relu activation. The output node had a softmax activation functoin with the adam optimizer and a categorical cross entropy loss function. For Q2, we had the same 2 hidden layers of 32 nodes but output nodes with sigmoid activation and binary cross entropy loss function.

The Multitask Keras DNN also had 2 hidden layers of 32 nodes, each with relu activation, 2 output nodes with softmax activation, Adam optmizer, and a categorical cross entropy loss function with a masked loss function for the Q2 task to ignore the unknown labels.

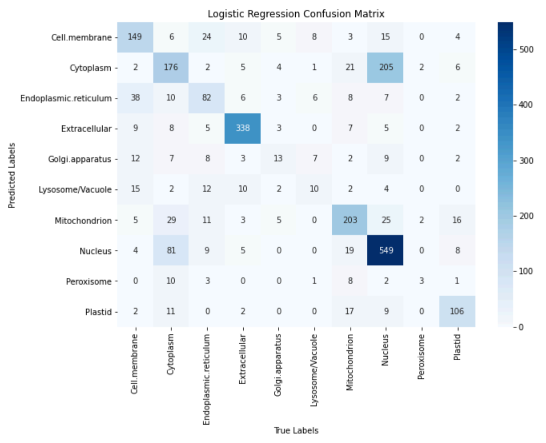
Logistic regression was fairly consistent between both the binary task as well as the 10 class task. The best Logistic regression model was optimized as well using GridSearchCV resulting in a C value of 1.0 and an l2 penalty. We will analyze and compare Logistic Regression and Keras DNN in the following sections.

5.2.3 Logistic Regression Q10 and Q2

The confusion matrix and classification report for Logistic regression for the 10 class task is shown in figure 12. Looking at the confusion matrix, we see



```
Logistic Regression Classification Report

                       precision    recall  f1-score   support

         Cell.membrane      0.63      0.67      0.65       224
             Cytoplasm      0.52      0.42      0.46       424
 Endoplasmic.reticulum      0.53      0.51      0.52       162
         Extracellular      0.88      0.90      0.89       377
       Golgi.apparatus      0.37      0.21      0.27        63
      Lysosome/Vacuole      0.30      0.18      0.22        57
         Mitochondrion      0.70      0.68      0.69       299
               Nucleus      0.66      0.81      0.73       675
            Peroxisome      0.43      0.11      0.17        28
               Plastid      0.72      0.72      0.72       147

             micro avg      0.66      0.66      0.66      2456
             macro avg      0.57      0.52      0.53      2456
          weighted avg      0.65      0.66      0.65      2456
```



Logistic Regression Confusion Matrix

Predicted Labels \ True Labels	Cell.membrane	Cytoplasm	Endoplasmic.reticulum	Extracellular	Golgi.apparatus	Lysosome/Vacuole	Mitochondrion	Nucleus	Peroxisome	Plastid
Cell.membrane	149	6	24	10	5	8	3	15	0	4
Cytoplasm	2	176	2	5	4	1	21	205	2	6
Endoplasmic.reticulum	38	10	82	6	3	6	8	7	0	2
Extracellular	9	8	5	338	3	0	7	5	0	2
Golgi.apparatus	12	7	8	3	13	7	2	9	0	2
Lysosome/Vacuole	15	2	12	10	2	10	2	4	0	0
Mitochondrion	5	29	11	3	5	0	203	25	2	16
Nucleus	4	81	9	5	0	0	19	549	0	8
Peroxisome	0	10	3	0	0	1	8	2	3	1
Plastid	2	11	0	2	0	0	17	9	0	106

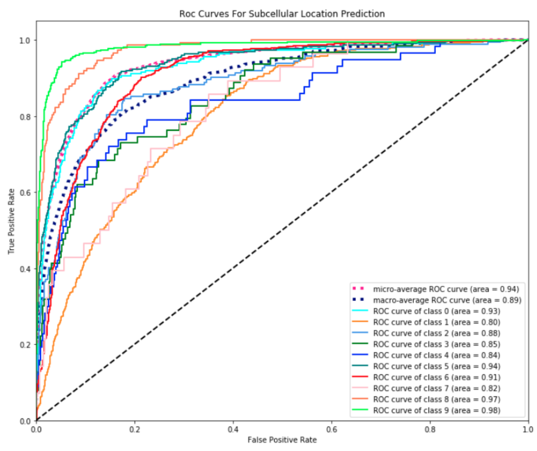
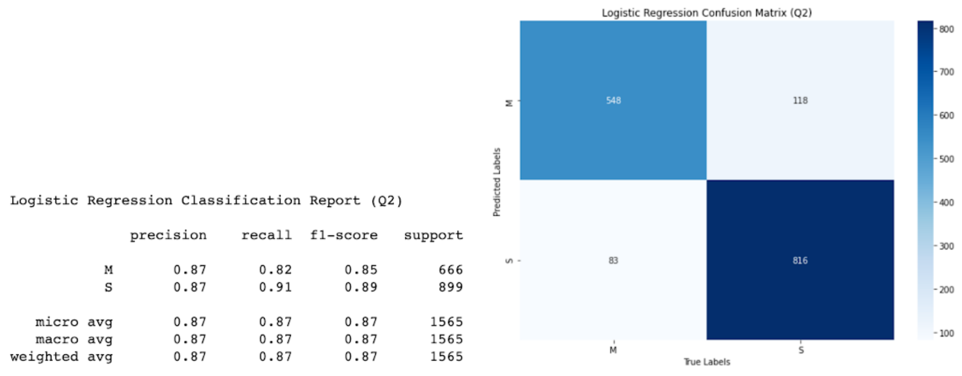


Figure 12: The classification report, confusion matrix and ROC curve for DeepLoc Logistic Regression Q10

that Cytoplasm and Nucleus were mislabeled with one another most often, and classes with low representation such as Golgi apparatus, Peroxisome, and Lysosome/Vacuole had very low precision and recall.

The ROC curve for the Logistic Regression model is shown in figure 12. A ROC curve is a performance measurement for classification problems and represent how much a model is able to distinguish between classes. A model that guesses randomly will produce an AUC (area under the curve) of 0.5, and directly follow the dashed line. Even though the AUC ranges from 0.80 to 0.98 across the different classes, this ROC curve indicates that even the worst performing classes which do not have as many samples significantly outperform random chance.

The confusion matrix and classifcation report for Logistic Regression for the binary task is shown in figure 13. Both the precision and recall is consistent between both classes, with an overall f1-score of 0.85 for membrane bound and 0.89 for soluble.

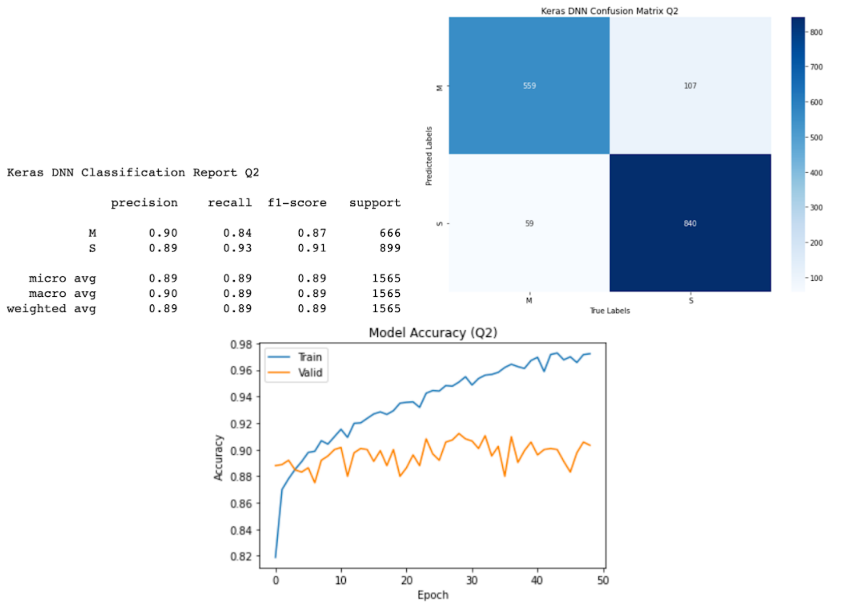


```
Logistic Regression Classification Report (Q2)

              precision    recall  f1-score   support

           M       0.87      0.82      0.85       666
           S       0.87      0.91      0.89       899

   micro avg       0.87      0.87      0.87      1565
   macro avg       0.87      0.87      0.87      1565
weighted avg       0.87      0.87      0.87      1565
```

Figure 13: The classification report and confusion matrix for DeepLoc Logistic Regression Q2

5.2.4 Keras DNN Q2

The confusion matrix and classification report for the Q2 Keras DNN is shown in figure 14. Over 50 training epochs the model was trained to achieve a slightly better f1 score as well as precision and recall for both membrane and soluble. The Keras model was better at predicting both classes.



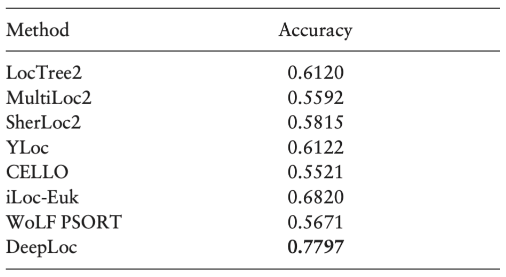
```
Keras DNN Classification Report Q2

              precision    recall  f1-score   support

           M       0.90      0.84      0.87       666
           S       0.89      0.93      0.91       899

   micro avg       0.89      0.89      0.89      1565
   macro avg       0.90      0.89      0.89      1565
weighted avg       0.89      0.89      0.89      1565
```

Figure 14: Classification report, training validation, and confusion matrix for DeepLoc Keras DNN

6 Evaluation and Findings

The benchmarks of existing models against the DeepLoc dataset are shown in figure 15. Looking at these results, we see that the DeepLoc researchers scored the highest with a 78 percent accuracy on the test set, whereas the next model iLoc-Euk achieves a 68 percent accuracy.

Looking at these results, we see that our support vector classifier outperformed 6 out of the 8 existing models on the DeepLoc test set, where our best model scored 67.5%. For the binary classification task, we achieved 89% with the Keras DNN, which is 3 percent lower than the result found by DeepLoc which was 92%. This result is substantial since we were able



Method	Accuracy
LocTree2	0.6120
MultiLoc2	0.5592
SherLoc2	0.5815
YLoc	0.6122
CELLO	0.5521
iLoc-Euk	0.6820
WoLF PSORT	0.5671
DeepLoc	**0.7797**

Figure 15: Evaluation metrics for DeepLoc dataset [14]

to achieve better results than most models benchmarked in DeepLoc with an off-the-shelf BERT model provided by TAPE. While the deep learning training didn't provide the best results, it still scored better than half of the other models described in DeepLoc.

Due to the success of the embeddings, we created a web interface for researchers to take protein sequence files and encode them with a desired pretrained model from TAPE such as BERT. This embedding step only requires a single GPU for training, and can be executed in a Google Colab notebook we also made available.

7 Scalability and Deployment

In order to achieve scalability, we leveraged AWS EC2's compute resources, specifically their Deep Learning AMIs, shown in figure 16. Additionally, we created a Dockerized flask app that could be deployed to EC2 using CloudFormation, a tool meant to deploy AWS services. Using BERT's pretrained model to embed input protein sequences for further modeling proved to be useful, both in visualization and testing of computationally simpler models other than the deep learning method. Based on this fact, we decided to create a web service deployable on EC2 that provides an interface for embedding raw fasta files, as well as visualizing embedded files using PCA in 2 and 3 dimensions.

The landing page of the flask app is shown in figure 17. The embedding endpoint leverages the Nvidia Cuda docker base image for full GPU support and scales to any number of GPUs you have on a given instance. It is essentially a wrapper over the TAPE embedding function, but provides an intuitive interface from a web browser to select models and other hyperparameters. The embedding endpoint outputs an npz file, which then can be input into the visualization endpoint with an accompanying labels file.

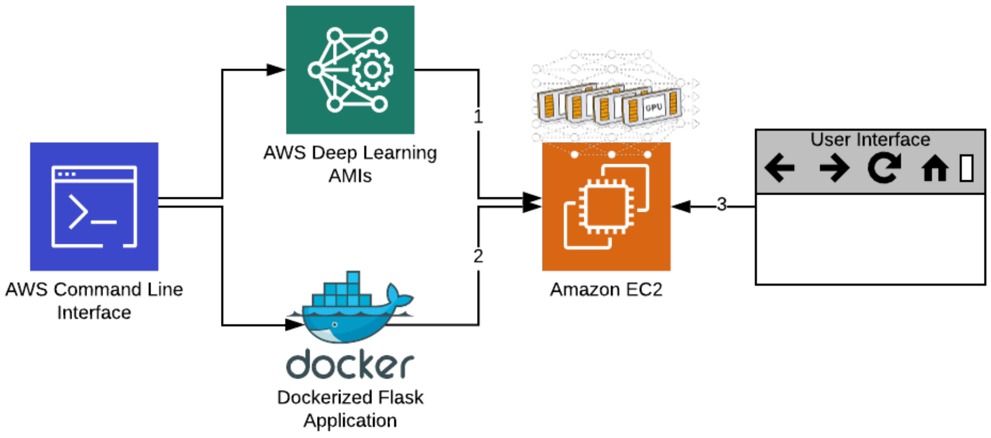
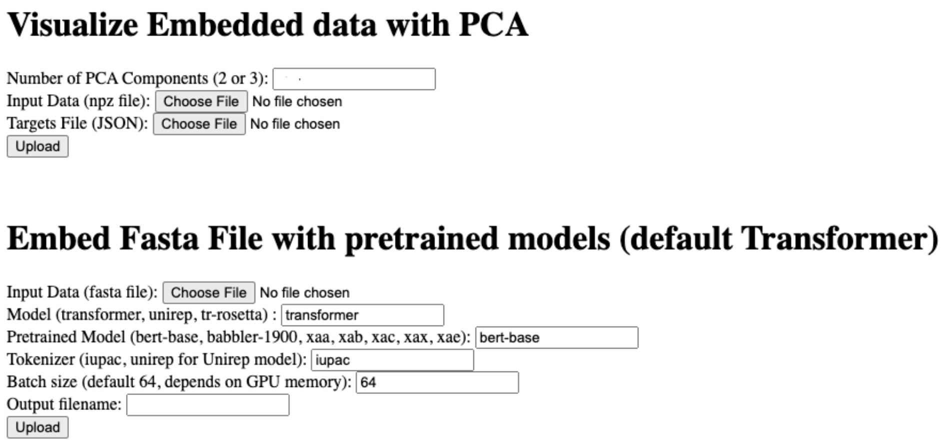


Figure 16: Architecture diagram of our data and compute infrastructure

Both the visualization and embedding endpoints can be exposed externally from EC2 and provided an easy interface for us to embed fasta files as well as visually inspect the embedding space.

The docker image isn't restricted to EC2, but can be run locally on a CPU as well, although embedding sequences using BERT or other pretrained models on CPU are very memory intensive.



Visualize Embedded data with PCA

Number of PCA Components (2 or 3):
Input Data (npz file): Choose File No file chosen
Targets File (JSON): Choose File No file chosen
Upload

Embed Fasta File with pretrained models (default Transformer)

Input Data (fasta file): Choose File No file chosen
Model (transformer, unirep, tr-rosetta) : transformer
Pretrained Model (bert-base, babbler-1900, xaa, xab, xac, xax, xae): bert-base
Tokenizer (iupac, unirep for Unirep model): iupac
Batch size (default 64, depends on GPU memory): 64
Output filename:
Upload

Figure 17: Landing page for the dockerized flask app

8 Conclusions

Our project aimed to investigate BERT and its application to protein sequences. Our methodology showed that using an off-the-shelf BERT model trained on a large amount of sequences provides an easy way to embed proteins to then be used in other modeling systems. When applied to protein sequences, BERT proves to be successful in protein prediction tasks defined in TAPE, as well as novel tasks shown by our tests with DeepLoc and subcellular location. These transformer based models leverage unsupervised pretraining of large amounts of data to understand the underlying structure and relationships of proteins. BERT and other transformer models continue to gain success in the NLP space, and continued research into training these models on protein sequences is necessary. For example, training BERT on a larger corpus of protein sequences may improve the quality of the embedding, thereby increasing the value of the methodology we have previously described.

Our results are substantial for two reasons. First this methodology was significantly less resource intensive and allowed us to train classifiers in frameworks such as sklearn, xgboost, and keras. Compared to deep learning frameworks that require expensive GPUs for large training, this pipeline makes modeling raw protein sequences with machine learning much more accessible to researchers. Additionally, while we knew that BERT was successful in tasks like fluorescence, stability, and secondary structure, we showed that its pretrained features contained knowledge for subcellular location, a novel task that it had never been exposed to.

The success of the off-the-shelf pretrained BERT model led us to create a web interface for embedding and visualization. The goal of this interface is to provide users an easy way to embed protein sequence files, then follow a similar method in using less computationally intensive frameworks like sklearn, xgboost, and keras. This service is built in Docker and can be deployed on any server to perform the embeddings and visualize the embedding space in PCA. We hope that this service can reduce the barrier to entry for most protein prediction tasks when it comes to pretraining.

In conclusion, our investigation into BERT proved insightful and meaningful. Further research into Transformer models using different corpuses and pretraining techniques can result in better quality embeddings, thereby increasing accuracy in downstream tasks. As more pretrained models are released, protein sequence modeling becomes more accessible to general researchers and machine learning engineers.

References

[1] Protein Structure: https://en.wikipedia.org/wiki/Protein_structure

[2] AlphaFold: https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery

[3] Natural Language Processing: https://en.wikipedia.org/wiki/Natural_language_processing

[4] https://bair.berkeley.edu/blog/2019/11/04/proteins/

[5] GLUE benchmarks: https://gluebenchmark.com/

[6] BERT github: https://github.com/google-research/bert

[7] http://jalammar.github.io/illustrated-bert/

[8] https://www.biorxiv.org/content/10.1101/622803v2

[9] https://www.biorxiv.org/content/10.1101/589333v1

[10] TAPE Repository: https://github.com/songlab-cal/tape

[11] TAPE: https://arxiv.org/abs/1906.08230

[12] Pfam: https://pfam.xfam.org/

[13] https://www. genecopoeia.com/product/subcellular_localization

[14] DeepLoc: https://academic.oup.com/bioinformatics/article/33/21/3387/3931857

[15] RCSB Protein Database: https://www.rcsb.org/

[16] Uniprot: https://www.uniprot.org/

[17] https://en.wikipedia.org/wiki/BLOSUM

9 Appendix

9.1 DSE MAS Knowledge Applied to Project

Throughout the course of the project we use much of the knowledge gained from the program. The lion's share of our project was carried out using Jupyter notebooks and the command line interface, which we learned about in depth in the first quarter of the program. Our PCA visualizations are driven by the materials learned in probability and statistics and carried out by the learnings gained from the course on data visualization. Machine learning was the backbone for our modeling and model evaluation tasks. Most importantly the program has taught us how to successfully implement and think about a data science project from end to end.

We would like to extend thanks and appreciation to the entire MAS faculty and team. It has been a pleasure and privilege to learn from you all. We wish you all the best and look forward to our alumni emails in the future.

9.2 Library Link and Citation

Dharma, Arjun; Dedhia, Rahil; Waldschmidt, Thomas B.; Rose, Peter (2020). Protein Embedding Analysis. In Data Science & Engineering Master of Advanced Study (DSE MAS) Capstone Projects. UC San Diego Library Digital Collections. https://doi.org/10.6075/J0KS6Q2H