Fixed-Length Protein Embeddings using Contextual Lenses

Amir Shanehsazzadeh
Harvard University
Cambridge, MA 02138
amirshanehsazzadeh@college.harvard.edu

David Belanger
Google Research
dbelanger@google.com

David Dohan
Google Research
ddohan@google.com

Abstract

The Basic Local Alignment Search Tool (BLAST) [4] is currently the most popular method for searching databases of biological sequences. BLAST compares sequences via similarity defined by a weighted edit distance, which results in it being computationally expensive. As opposed to working with edit distance, a vector similarity approach can be accelerated substantially using modern hardware or hashing techniques [11]. Such an approach would require fixed-length embeddings for biological sequences. There has been recent interest in learning fixed-length protein embeddings using deep learning models under the hypothesis that the hidden layers of supervised or semi-supervised models could produce potentially useful vector embeddings. We consider transformer (BERT [8]) protein language models that are pretrained on the TrEMBL data set [1] and learn fixed-length embeddings on top of them with contextual lenses [13]. The embeddings are trained to predict the family a protein belongs to for sequences in the Pfam database [9, 10]. We show that for nearest-neighbor family classification, pretraining offers a noticeable boost in performance and that the corresponding learned embeddings are competitive with BLAST. Furthermore, we show that the raw transformer embeddings, obtained via static pooling, do not perform well on nearest-neighbor family classification, which suggests that learning embeddings in a supervised manner via contextual lenses may be a compute-efficient alternative to fine-tuning.

1 Introduction

Our goal is to assess the viability of replacing BLAST's [4] dynamic programming approach with a vector similarity approach using fixed-length neural network embeddings. These embeddings will have to possess signal for protein features such as evolutionary relationships, structure, or function to be useful. Neural network generative models have become a popular tool for modeling proteins [17, 3, 5, 16, 21, 18, 7]. To apply these models to downstream tasks, embeddings are generated from the intermediate layers, generally with some kind of pooling. The above works have all shown evidence of transfer learning, where pretraining on unlabeled protein sequences results in improved downstream performance.

Contextual lensing is a general method of producing fixed-length vector embeddings, originally designed for sentences [13]. A contextual lens is a mapping that potentially includes learnable and self-attentive components and that applies some form of reduction, via pooling, over the sequence-length-dependent axis of a model's sequence representation. The motivation for this approach is to

Preprint. Under review.

avoid model fine-tuning, which is expensive and sensitive to hyperparameter choice, and instead freeze the encoder, which is generally a transformer or BERT [8] model, and learn a mapping on top of this encoder's embeddings. In Kiros et. al. [13], applying contextual lenses to BERT representations produced embeddings that were competitive with existing state-of-the-art results in NLP, which used models that required fine-tuning. TAPE's [16] "attention-weighted mean," which was used to learn fixed-length protein embeddings, can also be regarded as a lens.

In this paper, we show that a contextual lens can be used to learn robust embeddings from a pretrained transformer model that provide accurate protein family classification. To learn embeddings we treat the transformer as a frozen encoder and apply a learnable contextual lens. These learned embeddings are trained to predict the families of a group of sequences from the Pfam database that belong to a set of *lens families*. We then use this trained lens, along with the transformer encoder, to predict protein families belonging to a distinct set of families, which we refer to as *evaluation families*, by computing protein embeddings and using a nearest-neighbors classifier (1-NN). We find that this approach is competitive with 1-NN classification using BLAST's weighted edit distance, which suggests potential for a "Neural BLAST" that searches for sequences using embedding similarity. Our code is publicly available[1].

2 Neural BLAST

We represent proteins using an amino acid level encoding. A *lens architecture* is defined to be the combination of an encoder (one-hot encoding, CNN, RNN, transformer), 2-D per-residue representation to a 1-D fixed-length embedding, and a dense predictor layer. For contextual lenses we consider MeanPool, MaxPool, and LinearMaxPool (learnable linear transformation followed by MaxPool). More details can be found in Appendix A.

For training and evaluation we consider the task of Pfam [9, 10] family classification where we learn a mapping from a protein's primary sequence to its family. We take all sequences belonging to 10000 of the 17129 families, which we denote as *lens families*, and we do the same for 1000 disjoint families, which we denote as *evaluation families*. The lens families are used for *lens training* and the evaluation families are used for evaluating the embeddings with KNN.

2.1 Lens Training

Using the train-test split in [6] for the sequences belonging to the 10000 lens families, we train a lens architecture to directly perform family classification for these families. We measure the accuracy of this trained lens architecture on the test subset of the lens families, which we refer to as *Lens Accuracy*.

2.2 Evaluating Protein Embeddings for Nearest-Neighbor Retrieval

Our objective is to assess the viability of a Neural BLAST that searches sequences via vector similarity on neural network protein embeddings. To see if our model produces embeddings that are effective for such a task, we measure the performance of a KNN (1-NN) classifier on the embeddings of the proteins in the evaluation families. The idea here is that we have trained a model to classify the proteins belonging to the lens families, and we want to see if the embeddings it produces generalize and separate a distinct group of evaluation families.

We thus take the lens architecture, trained using lens training, and use it to compute fixed-length embeddings for the sequences belonging to the evaluation families. Again using the train-test split in [6], but this time for the sequences belonging to the 1000 evaluation families, we fit a 1-NN family classifier using the sequence embeddings. We fit using at most n samples per family from the train subset of the evaluation families and consider $n \in \{1, 5, 10, 50\}$ to see how well our embeddings perform with limited data. Finally, we take this 1-NN classifier and measure its accuracy on the test subset of the evaluation families. We refer to this metric as *n-Sample Test KNN Accuracy*. As a baseline we measure the accuracy of "BLAST-KNN" which is 1-NN using BLAST's weighted edit distance [4]. Note that BLAST-KNN can be thought of as simply returning the family of the top search hit in the database.

[1] https://github.com/googleinterns/protein-embedding-retrieval

3 Related Work

Bileschi et al. [6] develops ProtCNN, a deep, residual, dilated CNN trained for an analogous Pfam family classification task. This model, when trained on $\sim$1.1M samples corresponding to 17129 families using a random train-test split, achieves an error rate of 0.495% (accuracy of 99.505%). They ensemble 13 different ProtCNN models using majority voting to create ProtENN, which achieves an error rate of 0.159% (accuracy of 99.841%). These CNN models outperform variants of BLAST [4] models, which achieve $\sim$1.14-1.654% error rate. Senter et al. [19] considers the same high-level question as us of using protein embeddings for a vector similarity-based search. They train neural network models for classifying proteins from the RefSeq [14] database and find that such a training procedure, which is analogous to our lens training, produces embeddings that generalize to and separate unseen classes of proteins, allowing for effective nearest-neighbors classification. For more related works see Appendix B.

4 Models

Our lens architectures consist of three components: an encoder, a lens, and a dense predictor layer. The encoders we use are either CNN models or frozen transformer (BERT) models. For the CNN models we baseline a simple 2-layer CNN with kernel sizes 5 and feature sizes 1024 and also compare to the more complicated ProtCNN model [6], which is a deep, residual, dilated CNN. By frozen transformer we mean a transformer model that has frozen weights. We experiment with a 'Small' 2-layer transformer, a 'Medium' 6-layer transformer, and a 'Large' 36-layer transformer. We test pretrained versions of the transformer models as well. Pretraining is done on the TrEMBL data set [1]. The pretraining procedure involves truncation to a maximum length of 512 and training using the BERT objective [8] with the Adam optimizer [12]. The exact procedure is specified in Choromanski et al. [7] and their code is publicly available[2]. Our training procedure is described in Appendix C.

5 Results

Table 1: Best n-Sample Test KNN Accuracy for different model types and $n \in \{1, 5, 10, 50\}$.

	1-Sample	5-Sample	10-Sample	50-Sample
BLAST-KNN	0.861	0.978	0.991	0.996
2-Layer CNN	0.688	0.871	0.915	0.957
Transformer	0.779	0.921	0.957	0.981
Pretrained Transformer	**0.874**	0.974	0.985	0.996
ProtCNN [6]	**0.877**	**0.981**	**0.992**	0.994

In Table 1 we show the best (over random initializations and hyperparameter combinations) n-Sample Test KNN Accuracy for different model types. We bold any performance that beats the BLAST-KNN baseline. Figure 1 shows the top 10 n-Sample Test KNN Accuracy measurements vs Lens Accuracy for $n \in \{1, 50\}$ and for different model types ($n \in \{5, 10\}$ can be found in Figure 2 in Appendix D). Table 2 shows the best KNN accuracy results for the small transformer models with the MaxPool and LinearMaxPool lenses. For the performance of all transformer models with all lenses, see Table 3 in Appendix D. Recall that the MeanPool and MaxPool lenses are not learnable and so the corresponding embeddings are not directly influenced by the Pfam labels. For the randomly initialized transformer models, we took the maximum performance over 5 random seeds for MaxPool and MeanPool.

Table 1 shows that the pretrained transformer lens architectures outperform BLAST-KNN on 1-sample accuracy, but underperform as the number of samples increases. We also see that the ProtCNN [6] outperforms the lens architectures for any number of samples and beats BLAST-KNN for 1, 5, and 10 samples. Note that with the LinearMaxPool lens the pretrained transformer lens architectures significantly outperform the randomly initialized transformer lens architectures, especially when the error rate is considered. Figure 1 shows that with the LinearMaxPool lens, pretrained transformer lens

[2]https://github.com/google-research/google-research/tree/master/protein_lm

architectures outperform randomly initialized transformer lens architectures on both Lens Accuracy and n-Sample Test KNN Accuracy regardless of transformer size and relatively independently of random initialization or training hyperparameters. Table 2 shows that the pretrained transformer + LinearMaxPool lens architecture noticeably outperforms the randomly initialized transformer + LinearMaxpool lens architecture and that the non-learnable lenses considerably underperform.

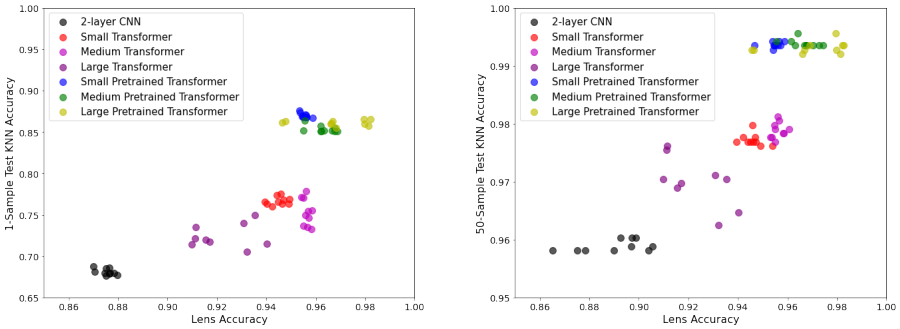

Figure 1: Top 10 n-Sample Test KNN Accuracy vs. Lens Accuracy for $n \in \{1, 50\}$. Transformer and 2-Layer CNN lens architectures all use the LinearMaxPool lens.

Table 2: Best n-Sample Test KNN Accuracy for small transformers and $n \in \{1, 5, 10, 50\}$. MeanPool and MaxPool lenses are static so their embeddings are not directly influenced by Pfam labels.

		1-Sample	5-Sample	10-Sample	50-Sample
Not Pretrained	MeanPool	0.398	0.639	0.719	0.830
	MaxPool	0.427	0.704	0.786	0.898
	LinearMaxPool	0.769	0.921	0.952	0.974
Pretrained	MeanPool	0.359	0.634	0.721	0.841
	MaxPool	0.419	0.683	0.770	0.881
	LinearMaxPool	**0.874**	0.969	0.980	0.993

6 Discussion

It is encouraging to see that neural networks, in particular pretrained transformer models and ProtCNN [6], can outperform the BLAST-KNN. This suggests that a Neural BLAST database search with comparable performance to BLAST but faster query time may be feasible. The results in Table 1 and Table 2 show some degree of transfer learning from the model pretraining as the pretrained transformers coupled with the LinearMaxPool lens outperform their non-pretrained counterparts regardless of model size or number of samples. Figure 1 not only supports this but also suggests that pretrainin offers an improvement in the distribution of performances. In Table 2 we see that both pretrained and non-pretrained models perform poorly with the static MeanPool and MaxPool lenses compared to the performance with the learnable LinearMaxPool lens. This indicates generalization from the lens training task and the corresponding performance boost is comparable to model fine-tuning, but requires substantially less compute.

Our results indicate that contextual lenses [13] offer a relatively inexpensive approach for the supervised learning of fixed-length protein embeddings that are useful for family classification. Additionally, we show that while the raw pretrained transformer embeddings, obtained via applying MaxPool or MeanPool, do not themselves perform well on nearest-neighbor family classification, the learnable LinearMaxPool lens is able to bring out the increased signal from the model pretraining to create more robust embeddings. This motivates us to suggest that contextual lenses be considered as an efficient alternative to model fine-tuning for downstream applications. We speculate that this framework can be used to learn embeddings designed for other downstream tasks, such as landscape or secondary structure prediction, and that these embeddings can be used to create a (potentially task-specific) Neural BLAST that performs sequence search via embedding similarity.

References

[1] https://www.uniprot.org/statistics/TrEMBL.

[2] Nomenclature and symbolism for amino acids and peptides (recommendations 1983). *Pure and Applied Chemistry*, 56(5):595–624, January 1984.

[3] Ethan C. Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M. Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, October 2019.

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[5] Tristan Bepler and Bonnie Berger. Learning protein sequence embeddings using information from structure. 2019.

[6] Maxwell L. Bileschi, David Belanger, Drew Bryant, Theo Sanderson, Brandon Carter, D. Sculley, Mark A. DePristo, and Lucy J. Colwell. Using deep learning to annotate the protein universe. May 2019.

[7] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, David Belanger, Lucy Colwell, and Adrian Weller. Masked language modeling for proteins via linearly scalable long-context transformers, 2020.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[9] Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, Erik L L Sonnhammer, Layla Hirsh, Lisanna Paladin, Damiano Piovesan, Silvio C E Tosatto, and Robert D Finn. The pfam protein families database in 2019. *Nucleic Acids Research*, 47(D1):D427–D432, October 2018.

[10] Robert D. Finn, Alex Bateman, Jody Clements, Penelope Coggill, Ruth Y. Eberhardt, Sean R. Eddy, Andreas Heger, Kirstie Hetherington, Liisa Holm, Jaina Mistry, Erik L. L. Sonnhammer, John Tate, and Marco Punta. Pfam: the protein families database. *Nucleic Acids Research*, 42(D1):D222–D230, November 2013.

[11] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[13] Jamie Kiros. Contextual lensing of universal sentence representations, 2020.

[14] Nuala A. OLeary, Mathew W. Wright, J. Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, Alexander Astashyn, Azat Badretdin, Yiming Bao, Olga Blinkova, Vyacheslav Brover, Vyacheslav Chetvernin, Jinna Choi, Eric Cox, Olga Ermolaeva, Catherine M. Farrell, Tamara Goldfarb, Tripti Gupta, Daniel Haft, Eneida Hatcher, Wratko Hlavina, Vinita S. Joardar, Vamsi K. Kodali, Wenjun Li, Donna Maglott, Patrick Masterson, Kelly M. McGarvey, Michael R. Murphy, Kathleen ONeill, Shashikant Pujar, Sanjida H. Rangwala, Daniel Rausch, Lillian D. Riddick, Conrad Schoch, Andrei Shkeda, Susan S. Storz, Hanzhen Sun, Francoise Thibaud-Nissen, Igor Tolstoy, Raymond E. Tully, Anjana R. Vatsan, Craig Wallin, David Webb, Wendy Wu, Melissa J. Landrum, Avi Kimchi, Tatiana Tatusova, Michael DiCuccio, Paul Kitts, Terence D. Murphy, and Kim D. Pruitt. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic Acids Research*, 44(D1):D733–D745, November 2015.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*.

[17] Adam J. Riesselman, John B. Ingraham, and Debora S. Marks. Deep generative models of genetic variation capture the effects of mutations. *Nature Methods*, 15(10):816–822, September 2018.

[18] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*, 2019.

[19] James K. Senter, Taylor M. Royalty, Andrew D. Steen, and Amir Sadovnik. Unaligned sequence similarity search using deep learning, 2019.

[20] B. E. Suzek, Y. Wang, H. Huang, P. B. McGarvey, and C. H. Wu and. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, November 2014.

[21] Jesse Vig, Ali Madani, Lav R. Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. Bertology meets biology: Interpreting attention in protein language models, 2020.

Appendix

A Background

A.1 Proteins

We consider proteins using only their primary structure, that is their amino acid sequence, with a 21-letter alphabet that includes the 20 standard amino acids [2] as well as a pad index. A length ℓ protein $a = a_1 a_2 \cdots a_\ell$ is thus modeled as a discrete sequence $x = (x_1, x_2, ..., x_\ell)$ with $x_i \in \{0, 1, ..., 19\}$ and potentially padded to $(x_1, x_2, ..., x_\ell, 20, 20, ..., 20)$.

A.2 Model Embeddings and Contextual Lenses

In general, an encoder model Φ (one-hot encoding, CNN, RNN, transformer) maps a protein x of length ℓ to a sequence-length-dependent array representation: $\Phi(x) \in \mathbb{R}^{n \times \ell}$ for some constant n. For learning fixed-length embeddings the goal is to learn another mapping Ψ such that $z = \Psi(\Phi(x)) \in \mathbb{R}^m$ with m being independent of ℓ and z being a useful protein embedding, in the sense of having strong signal for some downstream feature.

The simplest contextual lens [13] is pooling. This lens is non-learnable and involves either taking the average (MeanPool) or the maximum (MaxPool) over the length-dependent axis of $\Phi(x) \in \mathbb{R}^{n \times \ell}$. A learnable, but simple, contextual lens involves a single dense layer with weight matrix $W \in \mathbb{R}^{m \times n}$ and bias vector $b \in \mathbb{R}^m$ and a non-linear activation ϕ. Then we consider the array of transformed amino acid level embeddings

$$\Big[\phi(W\Phi(x)^{(i)} + b)\Big]_{i=1}^{\ell} \in \mathbb{R}^{m \times \ell}$$

and apply a pooling operation. If we apply MaxPool we call this operation LinearMaxPool and likewise for MeanPool. Note that we only use the ReLU activation for ϕ. A contextual lens that is both learnable and self-attentive involves a form of gated convolution. However, since we do not use this lens, we defer discussion of it to the original paper.

B Additional Related Work

We are unaware of other directly comparable works that attempt the same task, but machine learning techniques, specifically language models, have been used in a similar fashion to learn useful protein embeddings in [17, 3, 5, 16, 18, 21].

TAPE [16] proposes 5 downstream benchmark tasks and baselines a number of language models. Their language models, when pretrained on the Pfam database [9, 10], perform significantly better.

However, they did not find that performance on the language modeling task correlates strongly with downstream performance, which is a key motivating result from NLP.

RNN protein language models were developed by Alley et al. [3] and by Bepler et al. [5]. UniRep is a unidirectional multiplicative LSTM protein language model trained on UniRef50 [20]. UniRep learns embeddings that encode relevant protein information such as amino acid biochemistry, secondary structure, and evolutionary and functional information. Bepler et al. train a bidirectional LSTM that they pretrain on Pfam sequences [9, 10]. The learned embeddings are compared using a soft symmetric alignment and are useful for structure classification and transmembrane prediction.

Rives et al. [18] does similar work but trains varying size transformer language models on 250 million UniRef [20] sequences. Their transformer embeddings also encode amino acid biochemistry and achieve state-of-the-art performance on a number of downstream tasks such as contact, secondary structure, and remote homology. Additionally, they show that larger model size, which results in better protein language modeling performance, improves downstream performance. Vig et al. [21] consider transformer models as well and show that the attention mechanism itself, computed from pretrained protein BERT models, learns relevant features such as distance in 3-D, location of binding sites, and contact maps.

C Training Procedure

For lens training, we use the Adam optimizer [12] with variable learning rates and weight decays per lens architecture component, but with no learning rate warmup. Cross-entropy loss is used. Sequences are padded or truncated to a maximum length of 512 and padded components are zeroed out before pooling. We do not perform hyperparameter tuning but do a grid search over learning rates, weight decays, and random initialization seeds. For the transformer lens architectures the encoder learning rate and weight decays are always set to 0 since the transformer is frozen. Training hyperparameter combinations can be found in our GitHub repo[3]. For KNN classifiers we use scikit-learn's [15] KNN implementation.

D Supplementary Results

Table 3: Best n-Sample Test KNN Accuracy for all transformers and $n \in \{1, 5, 10, 50\}$. MeanPool and MaxPool lenses are static so their embeddings are not directly influenced by Pfam labels.

			1-Sample	5-Sample	10-Sample	50-Sample
Not Pretrained	MeanPool	Small	0.398	0.639	0.719	0.830
		Medium	0.423	0.663	0.742	0.848
		Large	0.412	0.655	0.740	0.848
	MaxPool	Small	0.427	0.704	0.786	0.898
		Medium	0.517	0.790	0.854	0.932
		Large	0.518	0.788	0.858	0.926
	LinearMaxPool	Small	0.769	0.921	0.952	0.974
		Medium	0.779	0.921	0.957	0.981
		Large	0.750	0.895	0.938	0.970
Pretrained	MeanPool	Small	0.359	0.634	0.721	0.841
		Medium	0.399	0.668	0.752	0.859
		Large	0.367	0.634	0.717	0.824
	MaxPool	Small	0.419	0.683	0.770	0.881
		Medium	0.395	0.647	0.739	0.864
		Large	0.389	0.656	0.736	0.851
	LinearMaxPool	Small	**0.874**	0.969	0.980	0.993
		Medium	**0.864**	0.968	0.985	0.994
		Large	**0.866**	0.974	0.985	0.994

[3] https://github.com/googleinterns/protein-embedding-retrieval/blob/master/params_combinations.json

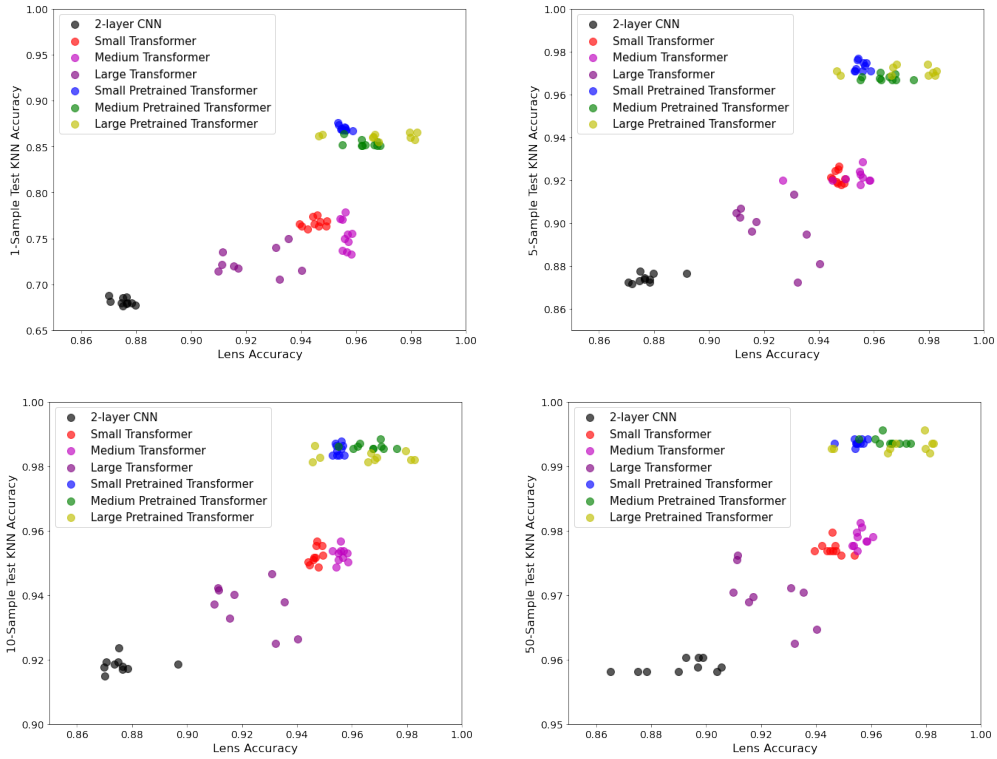

Figure 2: Top 10 n-Sample Test KNN Accuracy vs. Lens Accuracy for $n \in \{1, 5, 10, 50\}$. Transformer lens architectures all use the LinearMaxPool lens.