UNIVERSIDADE FEDERAL DE GOIÁS (UFG)

INSTITUTO DE INFORMÁTICA

PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO HENRIQUE FERNANDES DE CAMARGO

Future-Shot: Few-Shot Learning to tackle new labels on high-dimensional classification problems

Goiânia
2024

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[ ] Dissertação [x] Tese [ ] Outro*:_____

*No caso de mestrado/doutorado profissional, indique o formato do Trabalho de Conclusão de Curso, permitido no documento de área, correspondente ao programa de pós-graduação, orientado pela legislação vigente da CAPES.

Exemplos: Estudo de caso ou Revisão sistemática ou outros formatos.

2. Nome completo do autor

Fernando Henrique Fernandes de Camargo

3. Título do trabalho

Future-Shot: Few-Shot Learning to tackle new labels on high-dimensional classification problems

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Anderson Da Silva Soares**, **Professor do Magistério Superior**, em 12/03/2024, às 14:19, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Fernando Henrique Fernandes De Camargo**, **Discente**, em 12/03/2024, às 14:28, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4446280** e o código CRC **CFE3798B**.

FERNANDO HENRIQUE FERNANDES DE CAMARGO

Future-Shot: Few-Shot Learning para lidar com novos rótulos em problemas de classificação de alta dimensão

Tese apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás (UFG), como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Anderson da Silva Soares

Goiânia
2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Camargo, Fernando Henrique Fernandes de
Future-Shot: Few-Shot Learning to tackle new labels on high dimensional classification problems [manuscrito] / Fernando Henrique Fernandes de Camargo. - 2024.
LXXIV, 74 f.: il.

Orientador: Prof. Dr. Anderson da Silva Soares.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2024.
Bibliografia.
Inclui abreviaturas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. machine learning. 2. few shot learning. 3. metric learning. 4. triplet learning. 5. high-dimensional classification. I. Soares, Anderson da Silva, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE TESE

Ata Nº **05/2024** da sessão de Defesa de Tese de **Fernando Henrique Fernandes de Camargo** que confere o título de Doutor em **Ciência da Computação**, na área de concentração em **Ciência da Computação**.

Aos vinte e três dias do mês de fevereiro de dois mil e vinte e quatro, a partir das dez horas, via sistema de webconferência, realizou-se a sessão pública de Defesa de Tese intitulada **"Future-Shot: Few-Shot Learning to tackle new labels on high-dimensional classification problems"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Anderson da Silva Soares (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Arlindo Rodrigues Galvão Filho (INF/UFG), membro titular interno; Professor Doutor Roberto de Alencar Lotufo (Unicamp), membro titular externo; Professor Doutor Herman Martins Gomes (UFCG), membro titular externo; Professor Doutor Flávio Henrique Teles Vieira (EMC/UFG), membro titular interno. Durante a arguição os membros da banca **não fizeram** sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Tese tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Anderson da Silva Soares, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora e pelo Coordenador do Programa de Pós-Graduação em Ciência da Computação, Professor Doutor Fabrizzio Alphonsus Alves de Melo Nunes Soares, em substituição à assinatura do Professor Herman Martins Gomes, aos vinte e três dias do mês de fevereiro de dois mil e vinte e quatro.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Anderson Da Silva Soares**, **Professor do Magistério Superior**, em 23/02/2024, às 15:21, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Arlindo Rodrigues Galvao Filho**, **Professor do Magistério Superior**, em 23/02/2024, às 15:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Roberto de Alencar Lotufo**, **Usuário Externo**, em 23/02/2024, às 15:37, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Flavio Henrique Teles Vieira**, **Professor do Magistério Superior**, em 23/02/2024, às 17:06, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Henrique Fernandes De Camargo**, **Discente**, em 23/02/2024, às 19:01, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4403990** e o código CRC **ED7D7E76**.

Referência: Processo nº 23070.003006/2024-96 SEI nº 4403990

<I dedicate this work to my father Lucivaldo de Camargo, who, while he lived, was an example of a man for me. May God have him and may he always be proud of me.>

Acknowledgments

I acknowledge Amalgam and XNV for providing the necessary infrastructure and financial support to develop the first phase of the study and design the experiments. I acknowledge Igor M. Soares, Adriano Marques, and Oliver M. Crook for all the help with writing the first paper. I also acknowledge Anderson Soares for being my mentor and helping me shape this thesis.

Publications

Publications made during the development of this thesis:

1. Soares, I.M., Camargo, F.H.F., Marques, A. "***Ranking labs-of-origin for genetically engineered DNA using Metric Learning***". arXiv (2021). URL: https://arxiv.org/abs/2107.07878 [71].
2. Soares, I.M., Camargo, F.H.F., Marques, A. et al. "***Improving lab-of-origin prediction of genetically engineered plasmids via deep metric learning***". Nat Comput Sci 2, 253–264 (2022). URL: https://doi.org/10.1038/s43588-022-00234-z [104].
3. Soares, I.M., Camargo, F.H.F., Marques, A. et al. "***Using metric learning to identify the lab-of-origin of engineered DNA***". Nat Comput Sci 2, 296–297 (2022). URL: https://doi.org/10.1038/s43588-022-00240-1 [115].

Resumo

Camargo, F.H.F.. **Future-Shot: Few-Shot Learning to tackle new labels on high-dimensional classification problems**. Goiânia, 2024. 74p. Tese de Doutorado Relatório de Graduação. Programa de Pós Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás (UFG).

Esta tese introduz uma nova abordagem para enfrentar desafios de classificação multi-classe de alta dimensão, particularmente em ambientes dinâmicos onde surgem novas classes. Chamado de Future-Shot, o método emprega metric learning, especificamente triplet learning, para treinar um modelo capaz de gerar embeddings para pontos de dados e classes dentro de um espaço vetorial compartilhado. Isso facilita comparações eficientes de similaridade usando técnicas como k-nearest neighbors (KNN), permitindo integração de novas classes sem extenso treinamento. Testado em tarefas de previsão de laboratório de origem usando o conjunto de dados Addgene, o Future-Shot atinge a acurácia de top-10 de $90,39\%$, superando os métodos existentes. Notavelmente, em cenários de few-shot learning, ele atinge uma acurâcia de top-10 média de $81,2\%$ com apenas 30% dos dados para novas classes, demonstrando robustez e eficiência na adaptação às estruturas em que novas classes são inseridas com o passar do tempo.

Palavras–chave

few-shot learning, deep learning, metric learning, triplet learning, classificação em alta dimensão, predição de laboratório de origem

Abstract

Camargo, F.H.F.. **Future-Shot: Few-Shot Learning to tackle new labels on high-dimensional classification problems**. Goiânia, 2024. 74p. PhD. Thesis.Programa de Pós Graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás (UFG).

This thesis introduces a novel approach to address high-dimensional multiclass classification challenges, particularly in dynamic environments where new classes emerge. Named Future-Shot, the method employs metric learning, specifically triplet learning, to train a model capable of generating embeddings for both data points and classes within a shared vector space. This facilitates efficient similarity comparisons using techniques like k-nearest neighbors (KNN), enabling seamless integration of new classes without extensive retraining. Tested on lab-of-origin prediction tasks using the Addgene dataset, Future-Shot achieves top-10 accuracy of 90.39%, surpassing existing methods. Notably, in few-shot learning scenarios, it achieves an average top-10 accuracy of 81.2% with just 30% of the data for new classes, demonstrating robustness and efficiency in adapting to evolving class structures.

Keywords

few-shot learning, deep learning, metric learning, triplet learning, high-dimensional classification, lab-of-origin prediction

Contents

List of figures

List of tables

List of algorithms

List of acronyms

BPE Byte Pair Encoding. 14, 30, 38, 40, 41, 46, 47, 58, 61

CF Collaborative Filtering. 25
CNN Convolutional Neural Network. 13, 18, 21–25, 29, 38, 41, 48
CPU Central Processing Units. 22

DNA Deoxyribonucleic Acid. 28, 32, 35, 38, 43, 60

FSL Few-Shot Learning. 13, 17, 19–21, 26, 27, 35, 44, 47, 49, 50, 60, 62

GEA Genetic Engineering Attribution. 29, 30, 35, 47, 48, 60
GPU Graphical Processing Unit. 22

KNN K-Nearest Neighbors. 9, 10, 20, 31

LLM Large Language Model. 26

MLP Multi-Layer Perceptron. 21–23

NLP Natural Language Processing. 17, 60
NTI Normalized Token Importance. 13, 44, 54–58

OHE One-Hot Encoding. 14, 47

RNN Recurrent Neural Network. 21, 29, 30, 41

t-SNE T-distributed Stochastic Neighbor Embedding. 13, 52, 53

CHAPTER 1

Introduction

Machine learning has recently been an essential tool to automate tasks that have been mostly carried out by humans before. It is being used in an ever-growing number of applications using structured and unstructured data like images, audio, and text. However, there is one recurring problem always faced: training data size. There has been much research trying to identify the effects of training data size for different applications [133, 41, 8, 16, 32, 113]. Nevertheless, one thing is sure: machine learning models, especially deep learning ones, need a significant sample size to be accurate.

Since acquiring and curating data can be expensive, especially when specialized annotators are essential (like in the medical area), many researchers have been working on reducing the number of necessary data points. Data augmentation, for example, has been used a lot when dealing with images [99] and text [100] to virtually increase the size of the training dataset. Another crucial technique employed is tuning pre-trained models instead of starting from scratch. The oldest relevant example is using models pre-trained on ImageNet [18], which is a large image dataset. This trend also got into Natural Language Processing (NLP), with the most notable example being BERT [21], which is pre-trained in an unsupervised manner with a large corpus of text.

A more recent technique is becoming more common to deal with more extreme cases of data scarcity: Few-Shot Learning (FSL). The idea is to use prior knowledge to generalize to new tasks from a few samples [125]. It is being applied predominately in areas where many related data are available, like NLP. The most prominent example is the successful application of GPT-3 to tackle many NLP tasks using FSL [9].

Another common problem, especially with multiclass classification problems, is the high dimensionality of the number of classes [112]. The more classes one needs to deal with, the more complex the models are, and the more data is required to train them. After all, the model needs a reasonable sample size for each class, and if it is dealing with thousands or millions of classes, the data requirement can snowball. This is a known problem with multiple studies published [15] [85]. There is also an entire field of study dedicated to enhancing the efficiency of data collection, which can potentially alleviate the need for extensive data requirements [52] [37].

Moreover, some applications have an ever-growing number of classes (for example, identifying companies as new ones are being created). With traditional approaches, as new classes are added, acquiring more data and retraining the model to tackle it is necessary. A known problem with this approach is that multiclass models always classify examples into one of the known classes, sometimes even being very confident about their prediction. This can be solved with multiclass classification with rejection [74], but still requires retraining to deal with new classes.

Another different approach is stepping back from directly classifying those classes and instead training a model with an indirect objective. Such a model can be trained to learn to extract representations of each data point. Representation learning is a broad concept that involves learning a useful representation or encoding of the input data. The objective is to transform raw data into a feature space where relevant information is well-captured and can be easily utilized for downstream tasks. Once the raw data is transformed into a representation, it can be compared to other data points using clustering techniques.

This representation learning approach can be achieved by using metric learning. Metric learning focuses on learning a distance metric or a similarity measure between pairs of data points. The goal is often to optimize the embedding of data points in such a way that similar instances are brought closer together in the learned metric space, while dissimilar instances are pushed farther apart. This way, data points are represented by embeddings and can be compared for similarity. For classification purposes, instances of the same class can be clustered together in the vector space. When new data points need to be classified, they can be clustered with the training set, and the majority's class can be picked from the cluster. Alternatively, the model can learn embeddings for each class during training, which avoids the need to keep all the embeddings of the training set. Each data point embedding can be compared with each class embedding, picking the closer one in the vector space. In both cases, new classes can be added by extracting embeddings of a few data points of such classes. Such embeddings can either be stored for the clustering techniques or an average of them can be used as the new class embedding. After that, instances of previously unknown classes can be successfully classified.

Similar strategies have been adopted for recommender systems before, where the embeddings of users are compared to the embeddings of items. Anwaar et al. [5] extracted embeddings from the textual descriptions of movies and used them together with user ratings of such movies to generate user embeddings. The movies were then ranked according to the similarity between user embeddings and item embeddings. Lee et al. [57] took it one step further and proposed a deep content-user embedding model that combines the user-item interaction and music audio content. Their model uses a Convolutional Neural Network (CNN) to extract embeddings from audio while learning

an embedding layer for the users using interaction data. Both papers and several others in the area aim to deal with the cold-start problem, where new items that have no past interaction data need to be recommended, which is very similar to classification problems with an ever-growing number of classes.

There is also the application of FSL using embeddings. Liu et al. [63] use contrastive learning to train a few-shot embedding model to classify images. They train the model on a given set of categories and evaluate it using a different one. After that, they use a dataset with the target categories as support. New data points are matched with known ones by similarity and the label is picked. The main difference between this work and others similar to the problem at hand is that they focus on applying the model to a new task with a different set of labels, instead of adding new labels to the same task.

1.1 Motivation

While various techniques exist, the literature currently lacks a clear-cut framework for addressing high-dimensional multiclass classification with a growing number of classes. It is important to tackle this problem because the currently available approaches are not resource-efficient, usually requiring the retraining of the model, which usually requires specialized hardware and takes a long time. This research centers around addressing this gap. The question being answered is whether this problem can be solved effectively using a combination of embedding models and FSL. The utilization of FSL is particularly emphasized to handle novel classes within the same task effectively. The goal is to demonstrate that such a framework can be developed, showcasing its results in a study case and an extra benchmark.

1.2 Proposal

This thesis proposes a method to deal with high-dimensional multiclass classification problems where the number of classes can keep growing. Instead of training a traditional classification model that outputs probabilities for each class (typically using a softmax layer), the method works by training a model to learn representations of data points and classes in a shared vector space through metric learning techniques (more specifically triplet learning). This is similar to how a hybrid recommender system works: learning embeddings from the users while learning to extract embeddings from the content of the items. In this work's case, the model learns embeddings for each class while learning to extract embeddings from input data. Once this model is trained, the embedding of input data is compared to the class embeddings with a dot product, allowing the classes to be ranked from most to least similar. Such a technique is commonly known

as K-Nearest Neighbors (KNN) [17]. A new class can then be added by using the model to extract the embeddings of a few samples and taking the average of them to represent the new class. After that, the model can tackle this new class without further retraining. Since a similar strategy for adding classes has been seen on FSL literature [102], with the difference that they used it to tackle only new classes, the method proposed here has been named Future-Shot.

1.3 Organization

This work is divided into six chapters. Chapters 2 and 3 lay the foundation necessary to understand the proposed method. Chapter 4 presents the procedures, the proposed algorithm, and the main study case. Chapter 5 shows the results of applying the method to the main study case and benchmark dataset. For last, Chapter 6 presents the conclusions and future work.

CHAPTER 2

Background

This Chapter lays the fundamentals that are later referenced during this thesis. It presents deep learning in Section 2.1, convolutional neural networks in Section 2.2, using deep learning for classification problems in Section 2.3 and metric learning in Section 2.4. Few-Shot Learning (FSL) is later presented with its literature review in Chapter 3.

2.1 Deep learning

Despite the first idea of creating machine learning models that imitate how our brain works appearing in 1957, it took decades for it to evolve and become what is known as deep learning nowadays. The first iteration, known as Perceptron [88], consisted of a single layer of neurons that multiplies each input by a learned weight, sums the result, and then applies an activation function. Multiple methods were developed to train a Perceptron [127]. However, it could not solve problems as simple as an XOR [73], reducing the researchers' interest in neural networks. That might explain why it took 20 years for the current method used nowadays to be proposed: Backpropagation [89]. This method also allowed us to train Multi-Layer Perceptron (MLP), which could deal with non-linear problems [35].

MLP was a popular machine learning method but had its limitations. It was not possible to have many layers because of the vanishing gradient problem [6]. Furthermore, even though multiple variants of neural networks were invented, like Recurrent Neural Network (RNN) [40] and Convolutional Neural Network (CNN) [56], they all suffered from this problem. Training neural networks were also too demanding for the existing hardware, requiring very long periods of training.

It was in 2012 that Deep Learning became extremely relevant thanks to Krizhevsky [54] and his winning solution used in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [90]. This competition features a massive dataset of images intending to classify them into 1,000 classes. Most participants used traditional computer vision techniques to extract features from the images and feed them into traditional Machine Learning models. Instead, Krizhevsky used a CNN that he trained using

a Graphical Processing Unit (GPU). Since then, there has been a surge in the usage of Neural Networks, and many players have invested money and research to develop the necessary resources to make it easier to train them. Nvidia invested a lot in their GPUs, creating even server-grade ones to be used for Deep Learning, and invested a lot into the CUDA development (toolkit used to interact with their GPUs). There was also a surge in the development of frameworks like Tensorflow [67], and PyTorch [80]. This whole progress was possible because of techniques used to deal with the vanishing gradient problem and the realization that GPUs can train Neural Networks much more efficiently than a Central Processing Units (CPU).

2.2 Convolutional neural networks

Convolutional Neural Network (CNN) is a special class of neural networks commonly used to tackle computer vision problems. Unlike a MLP, which has all the neurons from one layer connected to all the neurons of the next layer (having to learn one weight for each connection), a CNN employs a technique for parameter sharing, severely reducing the number of weights. The main idea is that each layer contains multiple learned filters/kernels. Each kernel will go through the whole input (or previous layer output) and compute an output for each position, which is a mathematical operation known as convolution. Equation 2-1 presents the convolution operation over a two-dimensional space (typically representing images or signals). $f[x, y]$ and $g[x, y]$ are two functions defined over a two-dimensional space, n_1 and n_2 represent discrete coordinates that are used to shift the function g. The convolution operation computes the integral of the product of two functions, where one function is reflected and shifted over the other function across all possible shifts. The output of this operation is known as the activation map. Figure 2.1 shows this operation with a 2D input (which is very common for images).

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2] \tag{2-1}$$

When building a CNN architecture, each layer is defined by specifying the number of kernels, size of the kernels, stride (that controls if there will be any compression of the output), and if it will use padding (that helps guarantee the dimensions of the output will be the same as the input). Besides the convolutional layers, it is common to apply pooling operations to compress the activation maps and, last, an operation to turn the final activation maps into a vector to be fed into an MLP. With such an architecture, the Neural Network will learn kernels responsible for extracting features from the input. Lastly, the MLP will process the extracted features in a vector form to do the final task (classification, for example). Figure 2.2 shows a very simple CNN architecture without the MLP.

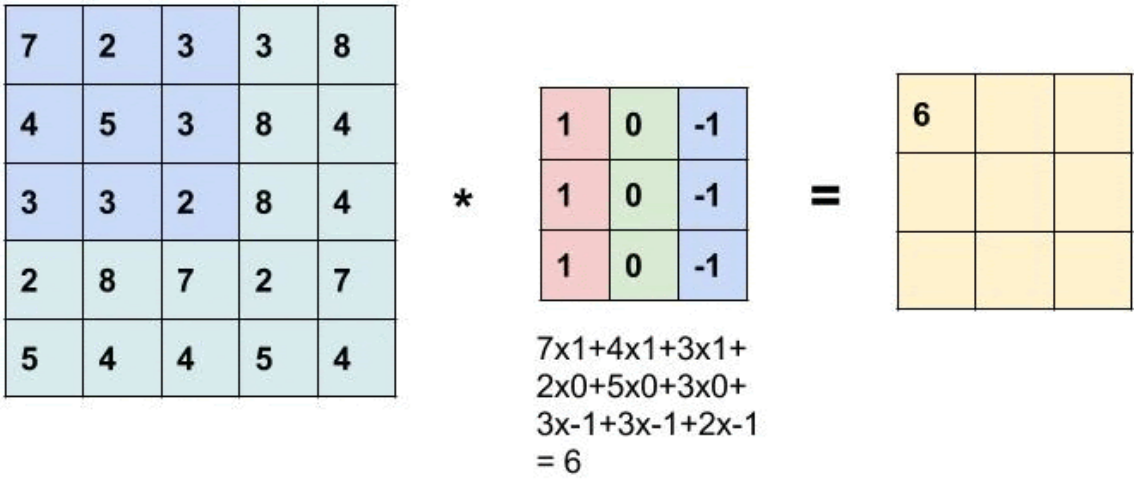


Figure 2.1: Convolution between a 2D input and a 2D kernel. The figure shows how the kernel (second matrix) shifts over the input (first matrix). The current position is highlighted in blue and the result of the first shift is shown in the third matrix. This exact computation is done for each position, filling in the remaining values.

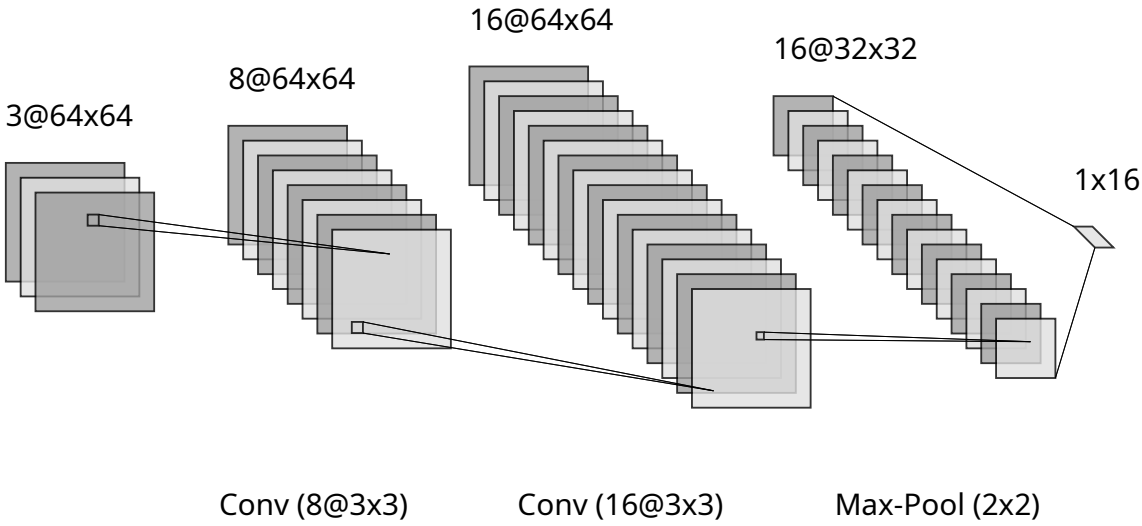


Figure 2.2: Simple CNN composed of an input (8 channels of size 128x128) that goes through one convolutional layer (8 filters of size 3x3 and padding), another convolutional layer (16 filters of size 3x3 and padding), one max pooling operation (size 2x2), and a max global pooling operation that reduces it into a vector of 16 dimensions, which can then be fed an MLP.

It is worth noting that both figures 2.1 and 2.2 deal with 2D inputs because CNNs are more commonly applied to images. Nevertheless, they can work with any dimension, including 1D (the case when dealing with textual data). One can also have parallel layers with different kernel sizes, allowing a significant variation of possible architectures.

2.3 Deep learning for classification problems

When using a Neural Network for a classification problem, one needs to pay attention to the final layer, its activation, and loss functions. If it is a binary classification, one expects the final layer to have a single neuron with a sigmoid activation function (defined in Equation 2-2, having z as the output of the neuron) to limit the output between 0.0 and 1.0, which is interpreted as a probability of the input being of this class. A binary cross-entropy loss function (defined in Equation 2-3) is used in this case. Binary cross-entropy measures the dissimilarity between the true binary labels y and the predicted probabilities p by computing the negative sum of the true label times the logarithm of the predicted probability plus the complement of the true label times the logarithm of the complement of the predicted probability. The final goal of this loss function is that the predicted probability p will be closer to 1.0 if the label applies and closer to 0.0 otherwise.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2-2}$$

$$\text{binary\_cross\_entropy} = -(y \log(p) + (1 - y) \log(1 - p)) \tag{2-3}$$

When dealing with a multiclass problem, the output is a vector of probabilities that sum to 1.0. In that case, the last layer has as many neurons as the number of classes and a softmax activation function (defined in Equation 2-4, having z_i as the output of each neuron) is applied after it. This activation function guarantees that all the outputs will be between 0.0 and 1.0, and the sum will be 1.0. To train the model, a multiclass cross-entropy loss function (defined in Equation 2-5) is used. Multiclass cross-entropy calculates the dissimilarity between the true multiclass labels $y_{o,c}$ and the predicted probabilities $p_{o,c}$ by summing over all classes c the negative product of the true label and the logarithm of the predicted probability for each class. The goal of this loss function and to make the predicted probability $p_{o,c}$ of the true label closer to 1.0 and the remaining probabilities of the other labels closer to 0.0.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2-4}$$

$$\text{multiclass\_cross\_entropy} = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2-5}$$

2.4 Introduction to deep metric learning

Metric learning aims to automate learning task-specific distance functions in a (weakly) supervised manner [55]. The goal is to learn a distance metric to compare two data points for their similarity or distance. One can use these learned distance functions to perform various tasks, including clustering, information retrieval, identification, and others. One of the most well-known applications is Face Recognition [64] [20] [119] [120] [118], in which a face is compared to a reference picture and they are matched if they are similar enough.

Most metric learning research deals with a single entity type. One exception is Collaborative Filtering (CF) [93], which learns the similarity between users and items. Even though people do not commonly recognize it as part of metric learning, they do arguably the same, the difference being that CF deals with two different entity types. There is even research applying triplet network (a well-known deep metric learning technique) [39] to CF [114] [60].

The last layer must have the desired embedding dimension when training a neural network for deep metric learning. Also, an activation function is not commonly used on this last layer. The rest of the model depends on the modality being tackled (e.g., CNN for images). One trains the model so that similar objects produce embeddings that are close in the vector space and the opposite for dissimilar objects. To do that, there are many different techniques, like contrastive representation learning [14], triplet learning [96], lifted structured loss [106], multi-class n-pair loss [105], and many others. This work uses triplet learning, which is explained in the context in Section 4.1.1.

CHAPTER 3

Literature review

3.1 Few-shot learning with deep metric learning

Machine learning algorithms typically require large quantities of training data. Sparse training data for each class can result in poor-quality predictions. The act of training machine learning algorithms to perform well in this task is known as Few-Shot Learning (FSL) [124, 25, 26]. Based on knowledge already acquired by a model trained in a similar task (with a prior set of classes), the FSL method aims to classify samples into a target set of classes. The model is provided with a support set composed of a few samples per class and it must be able to classify new examples (query set) into the new classes.

Many techniques have been explored in FSL literature. One that has been very prominent recently is using a Large Language Model (LLM) trained for question answering for FSL [9, 108, 95, 94, 29]. They aim to provide instructions and potentially a few examples to the LLM and ask it to classify a new data point. Instead of training a new model, a researcher would work on prompt engineering, which is the process of tuning the instructions passed to an LLM to improve the final results. This ended up creating a whole new research area about prompt engineering [98, 87, 58, 83, 33, 126], especially with the rise of GPT-3.5 and GPT-4. The main advantage of such an approach is how fast one can start prototyping and have a text classification ready. For some problems, it can give very accurate results but sometimes might serve just as a baseline for other fine-tuned models. It is also limited to text classification, not dealing with other forms of inputs, and could have a hard time dealing with high-dimensional classification problems because of the token size limit (number of words an LLM can process). Netherless, that is not the focus of this work.

The remaining studies are split between optimization-based and metric learning approaches. The first works by using meta-learning to train a model in a variety of tasks in such a way that it can later solve new learning tasks with a small number of samples. The model is retrained for this new task with a small number of gradient steps. Examples of this category include MAML [27], Reptile [75], and LEO [91]. Such techniques can

deal with the problem of FSL, but still require retraining the model, even if for just a small number of steps.

The metric learning approach attempts to train a model to learn an embedding and appropriate comparison metric. Some examples of metric learning approaches include Prototypical Networks [102], RelationNet [110], TADAM [79], MatchingNet [78], and infoPatch [63]. The model is then adapted to another task by extracting embeddings from a small number of examples and using clustering techniques for classification. With the more widespread availability of self-supervised foundation models pre-trained on large-scale external data, simpler approaches like Prototypical Networks have found new highs, achieving state-of-the-art in some benchmarks [43].

This work uses a technique that is closer to Prototypical Networks [102]. They opted for a simpler design in which their model learns a metric space in which classification can be performed by computing distances to prototype representations of each class. These prototype representations are computed by taking the average of the embeddings of the support set for each class. Classification is then performed for an embedded query point by finding the nearest class prototype. Figure 3.1 shows how it would look like in a 2-dimensional space, with each c_k being in the middle of each class cluster and x (an embedded query point) being classified into c_2 because of the smaller distance. This work use the same technique with the difference being that Prototypical Networks only deal with a new set of classes (from a new task), while this work merges the existing classes with the new ones, staying in the same task. New classes have their prototype representations computed and the existing ones have theirs learned during training.

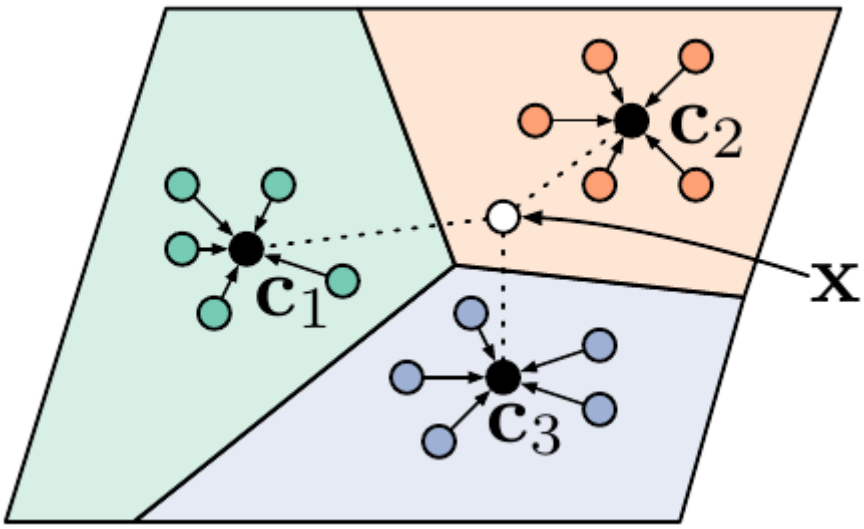


Figure 3.1: Prototypical networks in the few-shot scenario. Few-shot prototypes c_k are computed as the mean of embedded support examples for each class. A new input x is then embedded and then classified by computing the similarity between its embedding and all the prototypes c_k, picking the most similar class. Figure extracted from [102].

3.2 Genetic engineering attribution

3.2.1 Background

Plasmid sequences constitute fundamental components in the realm of genetic engineering and synthetic biology. These small, circular DNA molecules exist independently of the chromosomal DNA within a cell and often carry genes that confer specific advantageous traits to the host organism. Understanding the basics of plasmid sequences is pivotal for comprehending the intricacies of genetic engineering.

They are typically composed of double-stranded DNA, and their circular structure distinguishes them from the linear chromosomal DNA. Their size can vary significantly, ranging from a few thousand to several hundred thousand base pairs. Plasmids may contain essential genetic elements, such as an origin of replication, which enables them to replicate independently of the host chromosome.

Plasmids play a crucial role in genetic engineering as carriers of additional genetic material. Genetic engineers can introduce desired genes into plasmids, creating recombinant DNA molecules. These foreign genes may encode proteins with specific functions, such as resistance to antibiotics, the production of therapeutic proteins, or the enhancement of metabolic pathways in the host organism.

Computationally, a plasmid is represented as a string of characters, typically composed of the nucleotide bases adenine (A), thymine (T), guanine (G), and cytosine (C), denoted by the letters A, T, G, and C, respectively. This string of genetic information serves as a digital representation of the plasmid's DNA sequence. Each position in the string corresponds to a specific location along the plasmid's circular DNA structure. For example, a segment of the plasmid might be represented as "ATCGGTA," indicating the sequence of nucleotide bases at that particular region. This computational representation is fundamental for various bioinformatics applications, enabling researchers to analyze, compare, and manipulate plasmid sequences efficiently. Additionally, it forms the basis for advanced computational techniques, such as those employed in genetic engineering attribution, where distinct design signatures within these sequences can be identified to trace the origin of genetically engineered organisms.

Genetic engineering is conducted within specialized laboratories, often referred to as genetic engineering or biotechnology labs, equipped with advanced tools and expertise in molecular biology. These labs play a central role in the development and implementation of genetic modifications in organisms. As genetic engineers design and manipulate plasmid sequences within these labs, they inadvertently leave behind unique signatures in the form of specific nucleotide arrangements. These design signatures can be thought of as distinct patterns or markers that are introduced during the genetic engineering process. The inadvertent inclusion of these signatures provides a computational trail that can be

used for genetic engineering attribution, enabling the identification of the lab-of-origin for a given genetically modified organism. By analyzing the plasmid sequences, researchers can effectively trace the origin of the engineered organisms back to the specific laboratory where the genetic modifications took place. This attribution process not only aids in preventing plagiarism and ensuring responsible development but also establishes accountability within the genetic engineering landscape.

3.2.2 Lab-of-origin prediction of engineered DNA

Genetic engineering and synthetic biology represent rapidly advancing fields within biotechnology [23], where scientists possess the ability to modify organisms with unprecedented efficiency and sophistication. As these technologies become more prevalent, it becomes crucial to trace the origin of genetically engineered organisms back to their creators or the laboratories where they were developed. This process, known as Genetic Engineering Attribution (GEA), serves multiple purposes, including preventing plagiarism, promoting responsible development, ensuring proper credit for designers, and establishing accountability for genetic engineers.

The concept of GEA has gained significant traction in recent years, supported by advancements in attribution tools [2, 76, 122]. Essentially, when designing nucleic-acid sequences, engineers leave a unique "design signature" that can be detected using GEA methods. These methods have the potential to not only identify the lab of origin but also pinpoint the actual designer of a biological sequence. This capability enhances accountability within the genetic engineering landscape.

Various approaches to GEA have been proposed, with some focusing on predicting the lab of origin of plasmid sequences sourced from the Addgene data repository [45]. Notably, the performance of these approaches has seen significant progress, with top-10 accuracy improving from 70% to 85% in recent years [76, 1, 122]. Different methods, such as CNNs, RNNs, and pan-genome approaches, have been employed, indicating the versatility of GEA techniques.

Nielsen and Voigt [76] developed a deep learning model by applying CNNs. The network was trained on the Addgene plasmid dataset and independently verified in Alley et al. [1]. In brief, one can summarize their approach as follows. First, plasmid sequences were one-hot encoded, then used as input to the network composed of one convolutional layer of 128 filters, max-pooling operation, and two dense layers. While they showed it possible to use machine learning for this task, this seminal approach was hopeful and obtained an accuracy of only 48% and a top-10 accuracy of 70% in predicting the lab's origin.

More recently, Alley et al. [1] proposed deteRNNt, a recurrent neural network-

based model. The main insight of this approach was to treat the DNA sequence as a text problem, using techniques from the natural language processing field to extract features from the sequence. They tokenized the sequence using BPE, generating larger tokens and decreasing the sequence size. These tokens then served as input to a word embedding layer [69] followed by a RNN [62] [97]. The authors showed that their approach achieves 84.7% top-10 accuracy.

To serve as another baseline, BLAST [3] was applied to the test set of 18,817 samples. Despite being a relatively simple tool that employs no modern machine learning and finds similar local regions between sequences, it was able to predict source labs by achieving 76.9% top-10 accuracy in the conducted tests: outperforming the approach of Nielsen and Voigt [76].

PlasmidHawk [122], a recently launched tool, uses Plaster [121], a state-of-the-art pan-genome algorithm, to construct a synthetic plasmid resulting in a set of sequence fragments. It then aligns the original plasmid to the synthetic one and makes comparisons to match fragments with the plasmid. This method has so far outperformed other machine learning-based methods by obtaining 75.8% of accuracy and 85% of top-10 accuracy while employing no machine learning.

While these advancements are promising, there is room for further improvement. Existing approaches may face challenges in downstream tasks and require continuous retraining to adapt to new laboratories entering the scene. The ongoing refinement of GEA methods is crucial for ensuring the reliability and effectiveness of genetic engineering attribution in diverse and evolving biotechnological landscapes.

CHAPTER 4

Methodology

This research was conducted following several steps. First, the dataset of lab-of-origin prediction [45] (main study case) was obtained. Then, a regular classification model and the proposed model were trained, both using a similar model architecture to make them more comparable. They were evaluated for classification metrics (accuracy) and clustering, comparing their capabilities. Then, the proposed model is evaluated with few-shot learning experiments, showcasing how it can tackle new classes without retraining. Some interpretability techniques are also shown for both models. To further allow reproducibility and application of the method to other problems, a library was created and the method was applied to a benchmark dataset.

4.1 Proposed method

The proposed method uses deep metric learning [55], which is where one learns a distance function between objects. In this case, it is learning a similarity function between inputs and their classes. The result is a vector space in which an input embedding is close to its correct class embedding. This way, K-Nearest Neighbors (KNN) is applied to the input embedding and all the known class embeddings to find the closest classes, classifying this input. Whenever a new class becomes available, one can do few-shot learning by using a sample of inputs from this class, extracting the embeddings of this sample, and then taking the average to represent this new class. The model will then be able to classify this new class without further retraining. Because of the nature of the method, it was decided to name it future-shot.

To demonstrate that using deep metric learning indeed provides an advantage, a regular classifier with a similar architecture is developed to compare with the proposed approach. Both models share the backbone of the architecture to make the comparison fair. One can see the critical difference between them by looking at the final layers. The regular classification model's final layer is a dense layer with a softmax activation function, which returns a probability for each class. Instead, the future-shot approach has a dense layer that generates an input embedding. In parallel, there is an embedding layer

that learns the class embeddings. The main advantage of the future-shot approach is that it can extract embeddings of any input once it is trained. It allows one to cluster new inputs based on similar characteristics, rank or classify known classes, apply few-shot learning to deal with cold-starting, and even use the embeddings for other models.

The following Section presents the strategy to train the proposed model in detail.

4.1.1 Triplet learning with hard negative mining

To train the future-shot method, triplet networks [39] are used. This technique involves creating triplets (anchors, positives, negatives) as part of the model training process. The proposed model is anchored around the input; hence, they are the anchors in this approach. The positive object in this scenario is the correct class, while the negative object is another class. To be concrete suppose s_1 is a plasmid sequence made by the Church lab, then a possible triplet would be (s_1, Church lab, Voigt lab). Hence, this approach aims to generate embeddings in which the input is near its class and other inputs of the same class and far from other classes and their inputs. Figure 4.1 better illustrates this process while already considering the main study case (DNA as inputs and labs as classes).

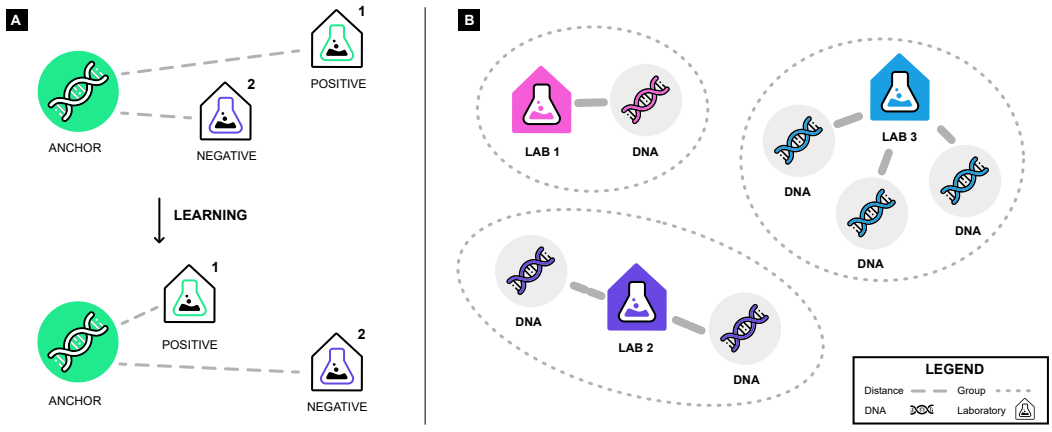


Figure 4.1: Triplet method illustration. The triplet is composed of an anchor (DNA), a positive (the lab-of-origin), and a negative example (another lab). **a** In the beginning, the anchor might be closer to the negative than it is to the positive. During training, the anchor and the positive are pulled towards each other while the negative is pushed away. **b** In the end, labs, and their DNA sequences will be nearer to each other, forming groups. Each group is represented by a different color. One can also expect labs and DNA sequences to be closer to other similar ones.

Normally, when training models using triplet learning, one has a single neural network that extracts embeddings from the input. Such neural network is run three times

to extract the embeddings from the anchor, positive and negative example, and then the triplet loss is computed using these three embeddings as shown in Equation 4-1. In essence, the loss is zero when the distance between the anchor and the positive example is smaller than the distance between the anchor and the negative example by at least the margin α. Otherwise, it penalizes the model based on how much the margin is violated.

$$L(a,p,n) = \max\{d(a_i,p_i) - d(a_i,n_i) + \alpha, 0\}$$
$$\text{where} \tag{4-1}$$
$$d(x_i,y_i) = \|\mathbf{x}_i - \mathbf{y}_i\|_p$$

However, as shown in Figure 4.2, Future-Shot is different than regular triplet learning in terms of architecture. Instead of running the neural network backbone three times, it is run only once for the input. The embeddings of the positive and negative examples instead come from an embedding layer, which is a matrix embeddings that allow gradient descent to flow and adjust the embeddings according to the loss function. This embedding layer contains one entry per known class and can later be updated to include more classes. Once this architecture is trained, the model operates by computing a dot product between the extracted embedding and the class embeddings matrix and taking the class with a higher similarity score.

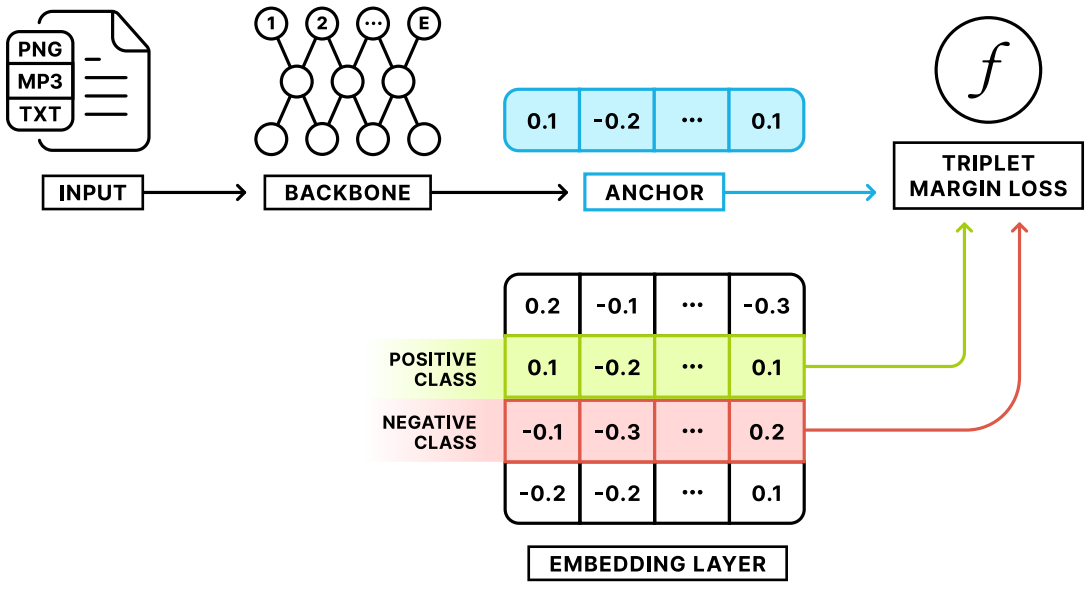


Figure 4.2: Architecture of Future-Shot. The input goes through a neural network backbone having a last linear layer of dimension E. In parallel, there is an embedding layer with dimensions (C, E), C being the number of classes and E being the embedding dimension. The positive and negative class indices come from the triplet generated from the dataset. The extracted embedding (anchor), the positive class embedding, and the negative class embedding are passed to triplet margin loss.

The labeled dataset is used to generate the triplets, providing anchors and positives. For the negatives, a technique known as hard negative mining [36] is used. It means that rather than choosing a random class as a negative example, the most challenging one given the current state of the embeddings (it can be different between different batches) is selected. Thus, in this case, it would be the nearest class to the input in the latent space.

One of the most challenging parts of this work was the implementation of the algorithm to mine the negative examples during training efficiently. A library called PyTorch Metric Learning [72] was first considered. They have hard negative mining implemented per batch (it does not take the whole dataset into account while finding the negative) and Cross-Batch Memory for Embedding Learning [123]. However, this library only supports a single entity type. Furthermore, the usage of an embedding layer gives easy access to the whole class embeddings. So, the approach was to re-implement this algorithm for the specific needs of this work (Algorithm 4.1). It is worth noting that it was implemented using tensors to make it as efficient as possible. The source code provides a PyTorch [80] implementation, and it should be straightforward to implement in Tensorflow [67] and other frameworks.

Algorithm 4.1: Algorithm for Hard Negative Mining using tensors. The shape of each tensor is added after each line as a comment, having B as batch size, E as embedding dimension, and C as the number of classes. It is worth noting that all the embeddings are L2 normalized.

Input:
class_indices (B,)
anchor_embeddings (B, C)
positive_embeddings (B, E)
Output: *negative_embeddings* (B, E)

all_class_indices ← indices of all classes repeated by B
// (B, C)
negative_classes_mask ← matrix of *True* values for every class
// (B, C)
negative_classes_mask[:, *class_indices*] ← *False*
// (B, C)
negative_class_indices ← *all_class_indices*[*negative_classes_mask*] reshaped
// (B, C-1)
negative_classes_embeddings ← *l2*(*class_embeddings*[*negative_class_indices*])
// (B, C-1, E)
anchor_similarities ← *dot*(*anchor_embeddings*, *negative_classes_embeddings*)
// (B, C-1)
hardest_negative_class_indices ← *argmax*(*anchor_similarities*)
// (B,)
return *negative_classes_embeddings*[:, *hardest_negative_class_indices*]
// (B, E)

4.1.2 Few-shot learning

The proposed method can straightforwardly be adapted to do FSL. Since the model learns to extract embeddings from inputs, one could add a new class by simply extracting the embeddings of a small sample and taking the average to generate the new class embedding. The new class can be added to the embedding layer without retraining the model; from then on, the model can start classifying this new class. Section 5.1.2 shows the results of this approach.

4.2 Case study

The goal of this work is to handle high-dimensional classification problems, especially the ones that one expects to have new classes appearing and need to deal with cold-start. Retraining the whole model again whenever new classes become available can be a huge burden, especially with models that might require special hardware and hours or even days to be trained. For that reason, the author aims to do few-shot learning [51, 26, 25, 124], tackling new classes without retraining the model. For those reasons, the main case study is lab-of-origin prediction (also known as GEA, as presented in Chapted 3.2), which has all those characteristics. This case study, the dataset used, and the models are presented in the following Sections.

4.2.1 Dataset

This work's approach is evaluated on a dataset from the Addgene repository [45], comprised of all plasmids deposited in the Addgene repository up to July 27th, 2018 – a total of 81,834 entries. The dataset includes a DNA sequence for each plasmid, along with metadata on growth strain, growth temperature, copy number, host species, bacterial resistance markers, and other selectable markers. Each of these categorical metadata fields is encoded as a series of one-hot feature groups as of Table 4.1.

In addition to the sequence and the above metadata fields, the raw dataset also contained unique sequence IDs and separate IDs designating the origin lab. Both sequence and lab IDs were obfuscated through 1:1 replacement with random alphanumeric strings. The number of plasmids deposited in the dataset by each lab was unbalanced, with many labs depositing one or a few sequences. To deal with this problem, Alley et al. [2] grouped labs with fewer than 10 data points into a single auxiliary category labeled "unknown engineered". It reduced the number of categories from 3751 (the number of labs) to 1314 (1313 unique labs + unknown engineered). In addition to issues with labs with small number of deposited plasmids, the dataset also contains "lineages" of plasmids (sequences derived by modifying other sequences in the dataset). If unmitigated, this introduces

Table 4.1: categorical metadata fields

Group	Features
Growth strain	growth_strain_ccdb_survival
	growth_strain_dh10b
	growth_strain_dh5alpha
	growth_strain_neb_stable
	growth_strain_other
	growth_strain_stbl3
	growth_strain_top10
	growth_strain_xl1_blue
Growth temperature	growth_temp_30
	growth_temp_37
	growth_temp_other
Copy number	copy_number_high_copy
	copy_number_low_copy
	copy_number_unknown
Host species	species_budding_yeast
	species_fly
	species_human
	species_mouse
	species_mustard_weed
	species_nematode
	species_other
	species_rat
	species_synthetic
	species_zebrafish
Bacterial resistance	bacterial_resistance_ampicillin
	bacterial_resistance_chloramphenicol
	bacterial_resistance_kanamycin
	bacterial_resistance_other
	bacterial_resistance_spectinomycin
Other selectable markers	selectable_markers_blasticidin
	selectable_markers_his3
	selectable_markers_hygromycin
	selectable_markers_leu2
	selectable_markers_neomycin
	selectable_markers_other
	selectable_markers_puromycin
	selectable_markers_trp1
	selectable_markers_ura3
	selectable_markers_zeocin

unintended correlations between the test and validation set. To overcome this, Alley et al. [2] inferred lineage networks among plasmids in the dataset based on information in the complete Addgene database acknowledging sequence contributions from other entries. Lineages were identified by searching for connected components within the network of entry-to-entry acknowledgments in the Addgene database. Refer to Alley et al. [2] for more details.

The data were partitioned into train, validation, and test sets following constraints that (i) every lab-of-origin has at least three data points in the test set, and (ii) all plasmids in a given lineage be assigned to a single dataset. Following the split, the training set contained 63,017 entries (77.0%); the validation set contained 7,466 entries (9.1%); and the test set contained 11,351 entries (13.9%). In order to evaluate the model's performance in this dataset, the test set was held out and the results were computed only once. This methodology is standard in machine learning tasks to avoid biasing researchers during the experiment. Therefore, the reported results refer to this hold-out set. The other tests performed during the construction of this work were evaluated using the validation set.

While training all models, a 4-fold cross-validation strategy [34] is performed for each experiment within the training set (63,017 entries). To be precise, for each hyperparameter setting, the data is split into k parts (being $k = 4$), using one of them to validate and the remaining to train, repeating this process k times. After that, each model is evaluated by taking the mean metrics for that experiment. This approach helps to avoid overfitting and improve generalization. It also enables one to ensemble the k models to improve generalization.

4.2.1.1 Grouping sequences by their Levenshtein distance

Genetic sequences from the same lab display large degrees of similarity. These sequences can make it easier to identify similar sequences in the training and validation sets. However, when training a machine learning algorithm, this may be perceived as data leakage between these sets [47], as the model does not need to learn to extract different features to identify such sequences. To ameliorate this issue, a more robust model was developed by grouping sequences from each lab based on their Levenshtein distance [7]. The Levenshtein formula used can be seen in equation 4-2.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{4-2}$$

After grouping, each laboratory has N groups of sequences. The dataset is then split, ensuring that there will be sequences from the same group only in training or validation, which means they will never be present in both sets simultaneously. This process is performed using Python and the python-Levenshtein library (https://github.com/ztane/python-Levenshtein). This approach complements the lineage-based strategy, which also avoids data leakage.

4.2.2 Models

Genetic sequences and phenotype information are inputs for the models. All sequences are processed to demonstrate each plasmid's characteristics better and improve the model's ability to identify patterns (Figure 4.3a). CNNs are used as the base model for the two approaches (future-shot and regular classifier). In this model, convolutional operations extract information in the sequence based on fixed kernel size. The models use convolutional structures with different kernel sizes in parallel, simulating the observation of sequences by pieces of different sizes. The information extracted by each structure is aggregated and added to the phenotype information and then assigned to one of the laboratories (Figure 4.3b).

The difference between the two approaches (regular classifier vs future-shot) is the training and output of the model. The standard approach treats the genetic attribution problem as a classification problem. A softmax layer is applied to determine the probability that the sequence belongs to each laboratory. On the other hand, the future-shot approach determines how far the features' representation of a new sequence is from the sequences cluster of a laboratory in the base. Smaller distances indicate more significant similarities between the features of a sequence and a lab-of-origin (Figure 4.3c).

Before training, all sequences from a given laboratory are grouped according to their Levenshtein distances. Sequences from the same group are not used in training and validation simultaneously, ensuring that sequences too similar do not cause leakage in training and overfit the model (Figure 4.3d). DNA sequences are compressed by the Byte Pair Encoding (BPE) algorithm [28] (more details in Section 4.2.2.1). It works by looking for common patterns in the sequence and unifying them into tokens, increasing the vocabulary while reducing the sequences' size (Figure 4.3e). Since plasmids are circular sequences, one can randomly shift the starting point of the sequence, increasing the number of training data (more details in Section 4.2.2.2). This method is performed "online" during sequence loading and network entry preparation. After all these processing steps, the sequence size is limited to 1000 characters to optimize the algorithm's training time and convergence capability (Figure 4.3f).

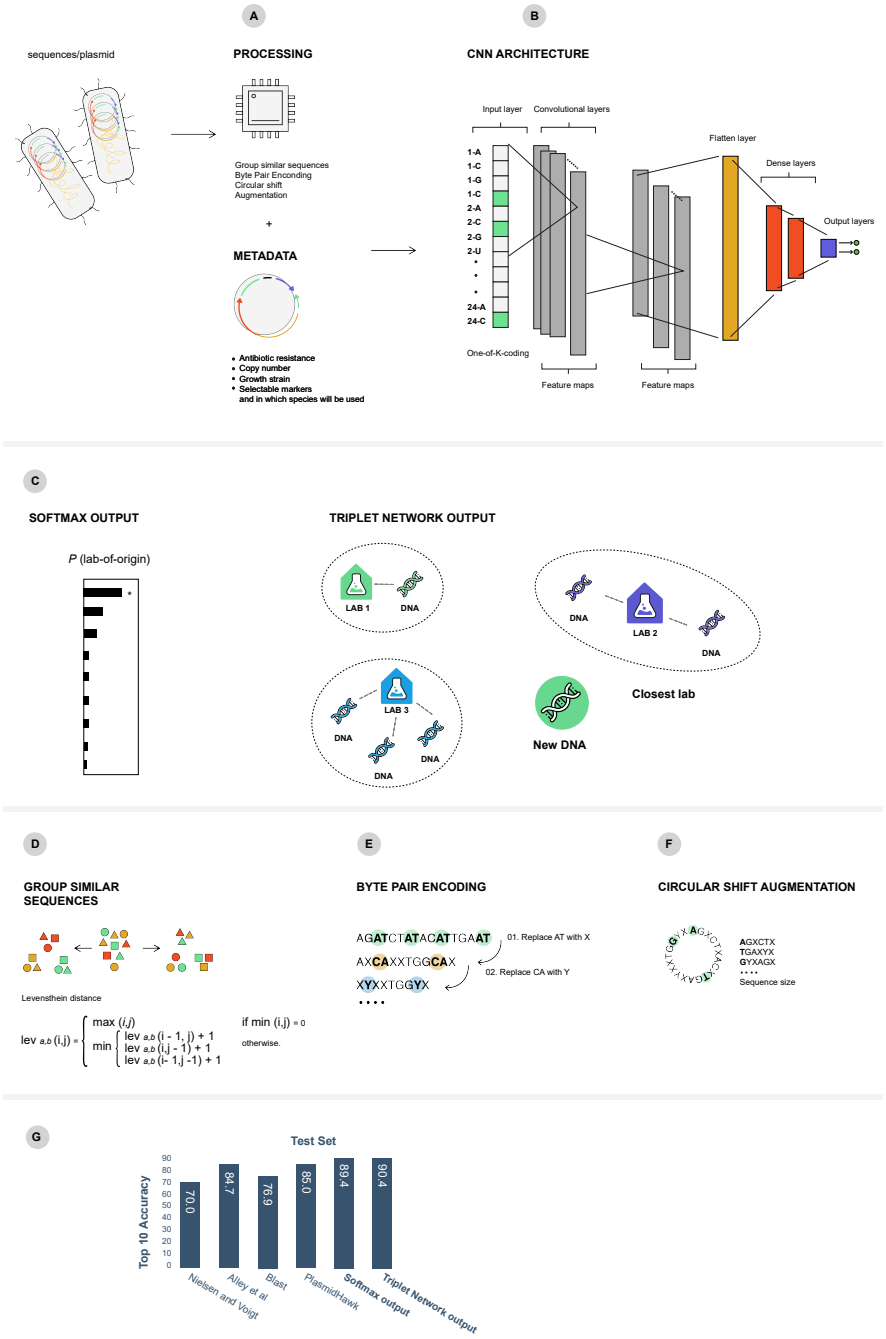


Figure 4.3: **(a)** Processing of the input. **(b)** CNN architecture used for both future-shot and regular classifier models. **(c)** The last layer of the regular classifier model has a softmax activation function, while the future-shot model learns the embeddings. **(d)** Process applied to group together similar sequences to avoid data leakage. **(e)** Byte Pair Encoding is used to reduce the sequence length and increase the vocabulary. **(f)** Circular Shift Augmentation is used to make the model shift invariant. **(g)** Results on the test set.

4.2.2.1 Byte pair encoding (BPE)

A commonly used method for encoding DNA sequences is the one-hot encoding scheme [65, 13]. However, following recent advances in engineered plasmid attribution, it is not worth to use the one-hot method to encode the sequence as it only encodes at the character level (nucleotides) and does not provide combination information. To confirm this hypothesis, experiments were performed changing the encoding technique between BPE and a one-hot encoding scheme with the same model, limiting the encoded sequence length. For the one-hot encoding scheme, 4000 characters were used as a limit, as this method does not tokenize characters and has worse results with lower limits. Section 5.1.1 shows the difference in results when using BPE and one-hot encoding. Using BPE is essential for better accuracy and enables a faster training time.

Further, to decrease the model's complexity, the sequence was limited to to 1000 tokens. As plasmids are usually extensive sequences, they can cause difficulties in model convergence and considerably increase training and inference time. To avoid these problems and find a good trade-off between limiting the sequence and not losing so much information, experiments were performed limiting the sequence to different lengths. It was found that limiting the sequence to 1000 tokens after applying BPE reduces the computational cost and improves results. The tests are shown in Section 5.1.1. The disadvantage is evident in the result and performance of the one-hot encoding scheme. This method does not allow one to limit the sequence, considerably increasing the training time, causing complexity in pattern recognition, and leading to lower accuracy. Although all experiments were performed with the triplet learning model, one can expect similar results for softmax since the issue is in the sequence's pre-processing.

4.2.2.2 Circular data augmentation

Machine learning models, especially deep learning models, are highly dependent on large amounts of data. One of the fundamental methods for adding variance to those models, increasing generalizability and reducing overfitting [12] is data augmentation [68]. Generally, data augmentation performs transformations on the sample, changing some characteristics. In this work, performing such transformations can be dangerous as it may modify some essential parts to assign the sequence to a lab-of-origin. However, it is possible to take advantage of the fact that plasmids are circular and create a circular shift data augmentation process. It contrasts with a reverse complement augmentation one might use. During training, one can show different versions of the same DNA sequence by shifting it circularly, as shown in Figure 4.3f. This approach helps the model understand that the same pattern can happen at different positions within the sequence, increasing the generalizability of the training.

Test time augmentation [70] was also performed to help improving the model's prediction capability. To perform this analysis, the model runs multiple times during inference. In tests performed, the model ran 10 times with different shifts. Each time the model sees a shifted version of the sequence and makes a prediction of the same example seen from different angles. The average of the output is taken as the final output (class probabilities for the classifier and embeddings for the proposed method).

4.2.2.3 CNN base architecture and training details

Both models are composed of a shallow convolutional neural network (CNN) with multiple kernels of different sizes, as proposed by Kim [48]. Figure 4.4 shows an example extracted from their paper for natural language. The idea of this architecture is that the model will learn multiple filters to extract features in an n-gram fashion. The conducted experiments used kernels with sizes from 2 to 12 to cover multiple lengths of sub-sequences that the model can recognize. It is worth noting that it is applied to the tokenized sequences, which already have multiple sub-sequences grouped into tokens by BPE. Compared to a deep CNN, this architecture cannot recombine the features in further layers but has fewer parameters to learn and a lower complexity, which helps to avoid overfitting. Compared to recurrent neural networks (RNN), this architecture can more easily deal with large sequences without worrying about vanishing gradients while having a much better computational performance. Transformers would also be an option, but they perform the best when pre-trained with a massive amount of data in a given domain.

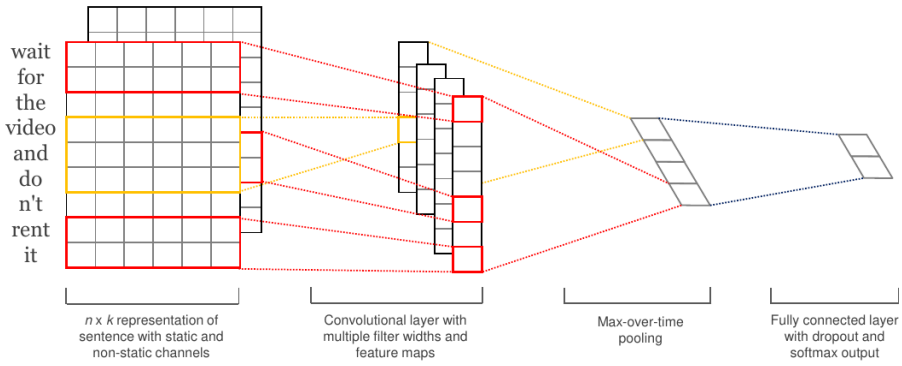


Figure 4.4: Text CNN model architecture with two channels for an example sentence. Figure extracted from [48].

This CNN extracts features from the sequence and concatenates them with the binary features provided in the dataset. The difference between the classification and triplet network models is in the final layers. The final base structure comprises an embedding layer, several convolutional layers in parallel with different kernel sizes, and a

custom dropout layer for regularization. The embedding layer has the shape of 1001x200, where 1001 is the vocabulary size, and 200 is the vector embedding dimension found empirically through hyperparameter optimization. Its purpose is to map each token into a 200-dimensional vector containing the features representation of that token [134]. For the convolutional layers, there is a total of 12 layers in parallel, where the first layer has kernel size 1, the second has kernel size 2 and it goes like this until the last layer has kernel size 12. A SeLU activation function [50] is applied to all convolutional layers followed by a max pooling operation. The features extracted by each of them are concatenated, obtaining the final representation with different windowing of the sequence. A custom Dropout Layer was also implemented. A standard Dropout Layer [107] randomly masks out parts of a tensor to regularize the neural network. However, the output would be too unstable if one did that on the embeddings before applying a similarity function. So, a layer that randomly masks out the same parts of all the embeddings involved before applying the similarity function was created. This approach was instrumental in regularizing the model.

The entire architecture and training of the models were developed using Python and Pytorch [80]. Although the training methodology differs between the two approaches, all the training details, such as optimizer, learning rate scheduler, and regularization techniques, remain the same. Adam [49] optimizer together with the One Cycle learning rate scheduler [101] were used. This scheduler was essential to achieve a better convergence in training, and its settings were a maximum learning rate of $1e^{-3}$ and cycle execution in 200 epochs. To regularize the model and prevent overfitting, a weight decay of $1e^{-5}$ was the best during training while the dropout rate was 20% with the custom dropout layer. Both models were trained using early stopping [31]. Early stopping is a technique used in machine learning to halt the training process of a model when its performance on a validation set starts to degrade, preventing overfitting. To find the best hyperparameters, multiple experiments were run using Bayesian optimization [103]. Bayesian optimization is a sequential model-based optimization technique that efficiently selects hyperparameters for machine learning algorithms by building a probabilistic model of the objective function and iteratively choosing parameter settings to maximize its expected improvement.

4.2.2.4 Cosine similarity

Cosine similarity was used as the metric to measure how similar the vectors were in the embeddings. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space, resulting in a value inside the range of -1 and 1, where -1 indicates opposite vectors and 1 indicates equal vectors. Given two non-zeros vectors of embeddings, A and B, with n dimensions the cosine similarity is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4-3}$$

Given the embedding extracted from a DNA sequence and the embedding of each lab, one can use cosine similarity to rank them from most similar to most dissimilar, picking the most similar to classify the sequence.

4.2.2.5 Interpreting the model

To visualize the mapped features of both models, two different approaches were taken since they are different models. First, all sequences from the validation set were inferred with the triplet network model, obtaining their embeddings. These embeddings were 200-dimensional, and they were reduced to 2 dimensions using *t*-SNE in scikit-learn project [81] with default parameters. For the classification model, the activations of the last hidden layer that maps features from all convolutional layers were extracted, before concatenating those features with the extra inputs (sequence metadata) and passing through the last layer which outputs logits. These hidden features were 3072-dimensional, reduced to 2 dimensions in the same way as embeddings. The visualization was prepared using matplotlib [44] and colored each point by the corresponding lab.

To analyze the influence of perturbations on model prediction, a specific plasmid was taken from the validation set and its sequence was randomly perturbated. As the process is random, ten experiments were performed for each number of mutations, ranging from 0 to 1000. To make these mutations, a random integer function was used to select the specific position to be changed and a random choice operation was performed to choose one of the five possible bases in the sequence (N, G, C, T, A). For each of the 10,100 runs, the position of the correct laboratory was taken in the model's prediction ranking. The results are shown in Section 5.1.4.

To determine which tokens are the most important within a sequence, a methodology similar to integrated gradients [109] was used. The integrated gradient is an interpretability technique for deep neural networks that finds the input features that contribute the most to the model prediction. First, gradients between model predictions with respect to the sequence embedding layer are computed, getting a matrix of gradients in the shape of 1001x200 (number of tokens x embedding dimension). Each gradient measures the relationship between the embedding weight and the output. For last, the absolute value element-wise is calculated and summed up in the second axis, generating a final vector of 1001 positions containing the summed importance of each token.

$$\text{token\_importance}_i(x) = \sum_{k=1}^{200} \| \frac{\partial F(x)}{\partial x_{ik}} \| \times \frac{1}{200} \tag{4-4}$$

where:

$F()$ = model's prediction function

$\frac{\partial F}{\partial x_{ik}}$ = gradient of model F's prediction function relative to each embedding feature x_{ik}

i = number of token

k = embedidng dimension position

After generating the token importance of each sequence in the validation set, the token importance of each laboratory was taken by averaging the token importance of all sequences from that laboratory. The visualization was prepared using matplotlib, and to better present the figure, the token importance values were normalized between 0 and 1, generating the Normalized Token Importance (NTI). To compare the NTI of a specific lab with a lab far from it, one can compute the cosine similarity between the analyzed lab embedding and all other lab embeddings by performing a dot product. The lowest value indicates the least similar laboratory.

4.3 Library and extra benchmark

To further highlight that the method developed in this thesis can be applied to other problems, an easy-to-use library was created. Such a library allows other researchers to apply the method to any classification problem and evaluate the results with the minimum code required. The library was then applied to an extra benchmark with a well-known dataset. Something similar to what was done in the case study was applied: training a regular classifier and future-shot model mainly using the same architecture, fine-tuning them, and comparing the results. However, the analysis was not as deep as the case study, as the idea was just to prove that the future-shot model can reach similar accuracy while allowing to tackle new classes with FSL.

4.3.1 Library

The library developed in this thesis is available at https://github.com/fernandocamargoai/future-shot. It is a Python library, designed around PyTorch [80] for the Deep Learning models, HuggingFace Datasets [59] to manage the loading and preprocessing of datasets, and PyTorch Lightning [24] to manage the training logic.

It is designed in a way that the user only needs to make a HuggingFace Dataset available (most well-known datasets are available through the HuggingFace Hub), possi-

bly implement a preprocessing function (following the API), implement a PyTorch model that outputs an embedding, and write a configuration file in YAML format [82] to put it all together. The whole training and evaluation of the regular classifier and the future-shot model are already implemented and readily available through command line commands. For further usage information, refer to the library documentation.

4.3.2 Dataset

Banking77 dataset [10] was chosen for the extra benchmark. It comprises 13,083 customer service queries labeled with 77 intents. It doesn't have as many labels as the case study, but it should be enough to demonstrate how the method can deal with new labels.

The dataset is already split between 10,003 examples for training and 3,080 examples for testing. The training set was further split into 8,002 examples for training and 2,001 for validation. The models are fine-tuned with the accuracy of the validation set and only the best model is evaluated with the test set to avoid data leakage in the fine-tuning.

4.3.3 Model

For the models, a pre-trained Sentence-BERT [86] (paraphrase-mpnet-base-v2) is used to extract embeddings (768 dimensions) from each document. It then goes through a Softmax layer in the regular classifier model while it's used directly in the future-shot model. This model was pre-trained to extract embeddings from sentences, which makes it perfect for tackling this problem. After training, the model is able to extract embeddings from customer service queries in a way that they can be grouped together according to their labels. The model also learns embeddings for each label and allows us to do few-shot learning.

CHAPTER 5

Results

5.1 Case study results

5.1.1 Classification

Future-shot model alongside the proposed training methodology improves the current state-of-the-art for attributing the lab-of-origin of an engineered DNA sequence, achieving 75.8% accuracy and 90.39% top-10 accuracy in a test set of 18,817 entries. For the classic approach of a classification model that predicts the input sequence's probability from any of the possible labs seen during training, this methodology surpasses all previous methods reaching 76.33% accuracy and 89.36% top-10 accuracy. These methods represent a $4-5$ absolute percentage improvement in performance over the current state-of-the-art in terms of top-10 accuracy[1]. At the same time, the future-shot approach improves over a simple softmax-based method using similar CNN architecture by 1 percentage point. Table 5.1 displays all the metrics.

Table 5.1: Accuracy and top-10 accuracy for previous works (different test sets) and both models. The hold-out set results were only computed after finding the best model using the validation set to avoid bias. This table also shows the results when not using the "unknown engineered" (UNK) category.

Model	Val Top-1 acc	Val Top-10 acc	Hold-out Top-1 acc	Hold-out Top-10 acc	All Top-1 acc	All-Top-10 acc
Nielsen and Voigt [76]	-	-	48.0	70.0	-	-
deteRNNt [1]	-	-	-	84.7	-	-
BLAST [3]	-	-	-	76.9	-	-
PlasmidHawk [122]	-	-	75.8	85.0	-	-
Future-Shot	78.15	91.88	74.23	90.39	75.78	**91.21**
Softmax	78.60	91.06	74.84	89.41	76.33	90.1
Future-Shot (no UNK)	**82.36**	**92.04**	**78.24**	**90.44**	**79.82**	91.05
Softmax (no UNK)	80.37	88.25	75.76	87.44	77.52	88.25

To further expand the results and understand the effects of different hyperparameters, Table 5.2 shows the difference in results when using BPE and ohe. In contrast, Table

[1]Worth noting that past studies might have different test sets, as the exact snapshot and split of the dataset is not available for previous studies.

5.3 shows the effects of the different sequence limits that were tried.

Table 5.2: Comparison between BPE and OHE. Accuracy and top-10 accuracy on the validation set, hold-out set, and the whole test set for the future-shot model using BPE and OHE.

Model	Val Top-1 acc	Val Top-10 acc	Hold-out Top-1 acc	Hold-out Top-10 acc	All Top-1 acc	All Top-10 acc
Future-Shot w/ BPE	**78.15**	**91.88**	**74.23**	**90.39**	**75.78**	**91.21**
Future-Shot w/ OHE	57.13	80.53	51.90	77.62	53.98	78.77

Table 5.3: Effect of limiting sequence size in both encoding methods. Results on the hold-out set, training time, and inference time for different encoding schemes and sequence size.

Encoding	Seq Limit Size	Hold-out Top 1 acc	Hold-out Top 10 acc	Training time	Iterations per second
BPE	500	70.39	88.97	**11h45min27s**	**20.16**
BPE	1000	**74.23**	**90.39**	15h14min35s	13.28
BPE	2000	69.00	86.76	18h22min35s	8.65
BPE	3000	69.20	86.82	21h18min51s	6.68
BPE	4000	68.56	86.70	1D0h16min58s	6.06
One-hot	1000	35.57	64.05	1D14h24min28s	4.73
One-hot	4000	51.90	77.62	3D03h31min40s	2.19

The differences in training and inference time between the softmax and future-shot models were also compared (Table 5.4). As one might expect, the softmax model has more iterations per second due to the lower complexity of calculating the loss function. The big difference appears in the total training time, showing that this model converges faster and therefore requires fewer training epochs when compared to the future-shot model. As GEA is generally an investigation process, the difference in inference time between the two models is not substantial. In conclusion, the future-shot model is more expensive to train and slower in production, but it is worth mentioning that it does not need any retraining to deal with new classes as the softmax counterpart.

Table 5.4: Training and inference time for future-shot model and softmax model.

Model	Training time	Iterations per second	Sequences per second (inference)
Softmax	**5h10min55s**	**14.94**	**25.39**
Future-Shot	15h14min35s	13.28	11.48

5.1.2 Few-shot learning

To test the proposed approach's ability to perform FSL, the following experiment was undertaken. The proposed method was trained 100 times, each time with 50 different labs being left out (all of their plasmids removed from the training set). For each lab that was left out, a random sample of N plasmids was picked to generate the embeddings

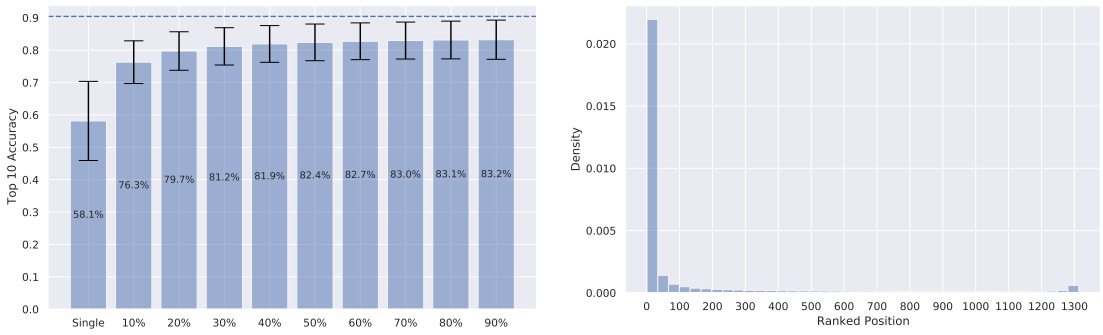
representing that lab. All remaining plasmids were used to evaluate the model. In the extreme case (also known as one-shot learning), a single plasmid was used to represent each lab and test with all the others. It is worth noting that the number of plasmids per lab varies a lot. There are labs in which 10% will be one or two, while in others, there will be hundreds of examples. For reference, the mean and standard deviation of each percentage are as follows: 10% (5 +- 18), 20% (9 +- 35), 30% (13 +- 53), 40% (17 +- 71), 50% (22 +- 88), 60% (26 +- 106), 70% (31 +- 124), 80% (35 +- 142), 90% (39 +- 159). Figure 5.1(a) shows the metrics' mean and standard deviation of the model's capability with different sample sizes used to represent each lab. As expected, the larger the sample, the greater the top-10 accuracy. However, there are diminishing returns as one increases the sample size. It shows that, in general, only a few representative examples are needed for high-accuracy predictions. This approach obtains better top-10 accuracy than the previously published CNN approach of Nielsen and Voigt [76] while only using 10% of the training data without requiring the retraining of the model.

Figure 5.1(b) and Table 5.5 show the rank of the lab-of-origin when taking a single plasmid to represent each lab. As can be seen, in most cases (79%), a single plasmid is enough to rank it at least in the top 100 (given the labs in the dataset). It is worth noting that the median (50% of the cases) ranked position was fifth. The variance can be high because their chosen plasmid may not represent the lab well. One can also observe that to make sure the actual lab is within the selected sample with a probability of 0.9, it is necessary to include around 685 labs. In other words, an analyst can rule out around 50% of the labs as the origin with the confidence of 90% when only a single plasmid is available for that lab. Furthermore, it demonstrates that if an analyst can pick a single representative plasmid of a lab using domain knowledge, this approach has an excellent chance to attribute an unknown plasmid to that lab without needing to retrain the model.

Table 5.5: Position of the correct lab-of-origin using one-shot learning. Considering the experiments in which a single plasmid was picked to represent each lab, this Table shows the ranked position of the lab for each percentage of the cases. For example, 50% of the time, the lab-of-origin was ranked 7th or lower. It is worth noting that a random plasmid for each lab was picked, and this experiment is run multiple times. If an analyst could select a plasmid that they believe represents the lab well, one could expect an even better performance.

Quantile	Ranked Position
50%	7 (top 0.4%)
60%	17 (top 1.2%)
70%	37 (top 2.7%)
80%	180 (top 13.6%)
90%	685 (top 52%)

So far, all the work on GEA ignores labs with few samples, grouping them into a



(a) *FSL Top-10 Accuracy*

(b) *Ranked position of correct lab using One-Shot Learning*

Figure 5.1: FSL results. **(a)** Each bar shows the mean and standard deviation of Top 10 Accuracy across all the FSL experiments (1,000 runs). The first bar (Single) is the extreme case where a single plasmid is picked to represent the lab. The rest of them refer to picking a percentage of the plasmids to represent the lab and using the rest to evaluate. Lastly, the dashed line refers to the Top 10 Accuracy when retraining the model. **(b)** The histogram presents the ranked position of the lab-of-origin when using a single plasmid to represent it.

single class called "unknown engineered". It was thus decided to examine this meta-class using the FSL approach. Figure 5.2 shows the extreme case of applying FSL to those rarest labs. Those labs (2-10 plasmids) were removed from the training data and then had the model evaluated on them with FSL. One can observe a marked drop in performance when examining these classes. However, it was expected given the small amount of data used for prediction in FSL (1 to 9 plasmids) and the vast number of available classes that can predicted.

5.1.3 Clustering

Clustering is one common application that can provide insights into the relationship of different plasmids and labs[46, 129, 77]. Many labs will share information about design techniques, have been mentored or trained in another lab, or directly collaborate. However, those relationships and similarities are not always known to us. Even though these similarities are not directly apparent through labs' embeddings, one can also examine which plasmid sequences of a particular lab are more similar to those of other labs if necessary. Once embeddings are obtained, one can apply any of the well-known clustering techniques. Figures 5.3, 5.4, 5.5, and 5.6 showcase common clustering techniques that an analyst could apply to the labs. It is possible due to the future-shot approach, which was not part of any previous work. Figure 5.3 shows the application of the well-known Elbow Method [111] to find a good number of clusters while avoiding overfitting. It works by plotting the explained variation by the number of clusters and picking the elbow of the

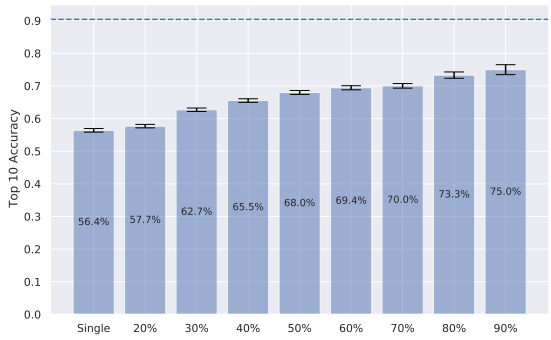


Figure 5.2: FSL Top-10 Accuracy for unknown labs (between 2 and 10 plasmids). Same structure of Figure 5.1 but applied only to the labs that were initially grouped for having too few plasmids (between 2 and 10). 1,000 runs were done to compute the mean and standard deviation.

curve. As can be seen, one could group the labs within various numbers of clusters, but 17 seems to be the optimal number (with a slight margin), which is why this number is used in both Figures 5.5 and 5.6. Figure 5.6 demonstrates a vector relationship between the laboratories, making it possible to group them to create a hierarchy using Agglomerative Clustering [132] (Figure 5.4) and reduce the search space. It is worth noting that an analyst would apply these techniques with a specific goal in mind, like studying the relationship of a target lab with others. For example, these clusters may link researchers geographically or collaboratively, which may help genetic forensics identify groups of labs even if a single lab is not identifiable.

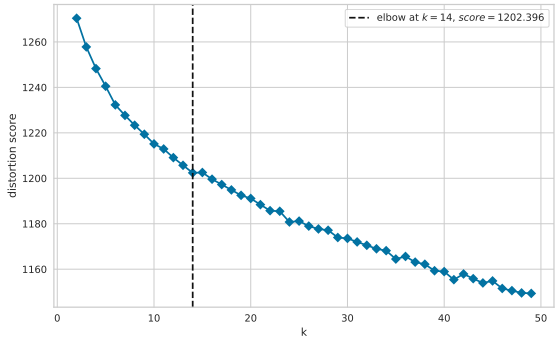


Figure 5.3: Distortion Score Elbow for KMeans Clustering.

5.1.4 Interpretability and robustness

Interpreting deep learning models gives one valuable information, such as understanding how the model works and the relative importance of features within the data. It can also reveal why some approaches work better than others, and this can be used to improve the model further. However, interpretation techniques for deep learning are still

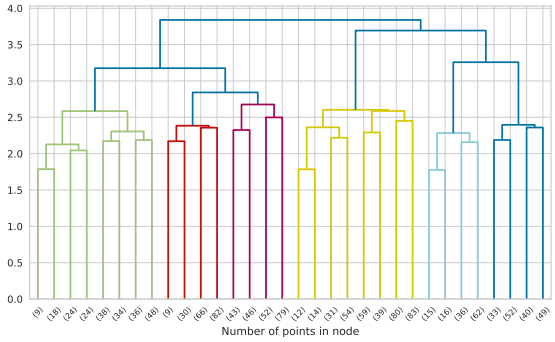


Figure 5.4: Hierarchical Clustering Dendrogram (4 levels).

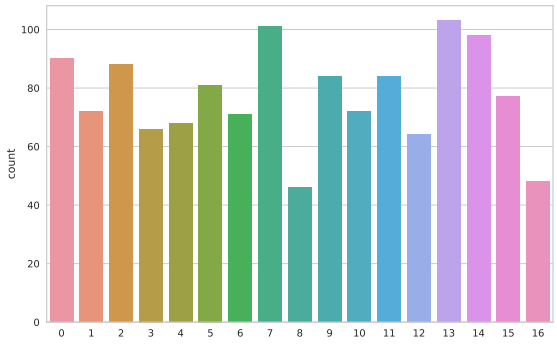


Figure 5.5: Labs by cluster (17 clusters). Using the optimal number of 17 clusters (given by the Elbow Method), the number of labs per cluster is obtained. The clusters are generally very similar in size, but some substantially differ from the average.

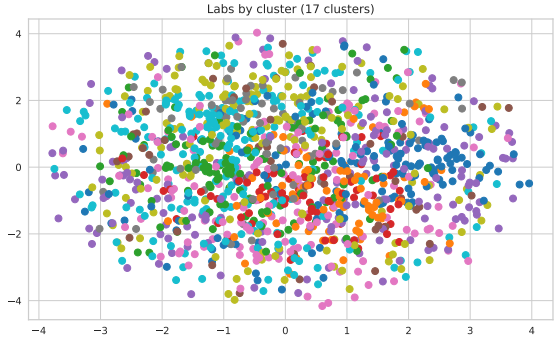
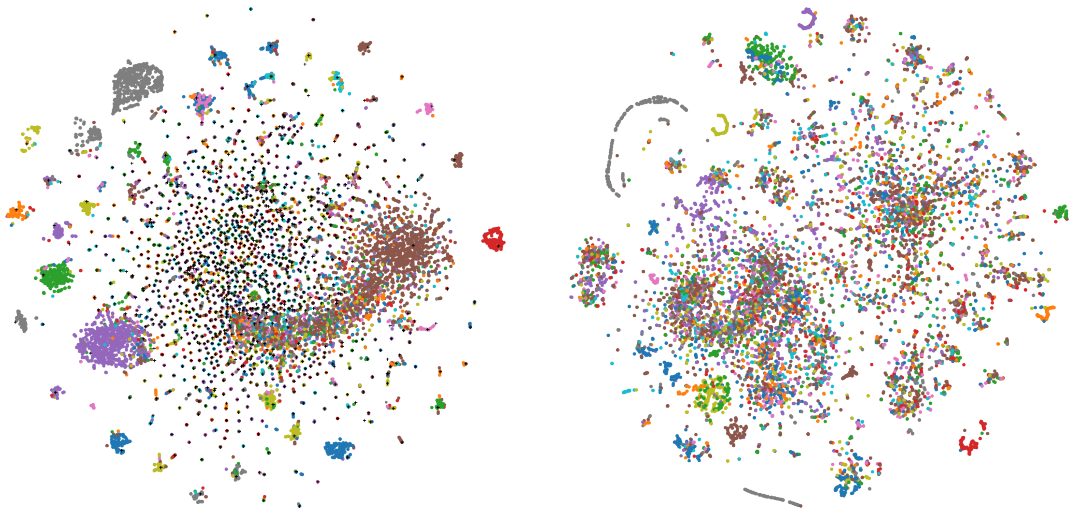


Figure 5.6: Embeddings analysis. Shows the labs (using the same number given by Elbow Method) in a 2D space (x and y coordinates) after compressing information using t-SNE [38]. The colors represent their clusters.

naive and are an area of active research [11, 22]. In this work, the focus was on understanding the differences between the future-shot and a conventional classification model, how robust the proposed model is when performing point mutation, and, most importantly, which tokens (parts of the sequence) are critical for identifying a lab.

One can start by visualizing the differences between the space of features mapped between the two models. For the future-shot model, this space is the sequence embeddings. While for the softmax model, the output of the 3072-dimensional last hidden layer is taken. This layer is the concatenation between all convolutional layers and contains all the features used by the model. The two multi-dimensional vectors are reduced to 2D space using T-distributed Stochastic Neighbor Embedding (t-SNE) [66]. As mentioned in Alley et al. [1], the model is more accurate when the plasmid features are more separable in the latent (unobserved) space. Figure 5.7(a) shows that the future-shot model has better-defined clusters. It is noticeable that the DNA sequences group together very well with their lab-of-origin. Some labs display very similar DNA sequences, while others are dispersed. It showcases the differences between large and small labs. Figure 5.7(b) shows the clustering with the softmax model. Although clusters can be seen in this feature map, they are not as well defined as in the future-shot model.



(a) *t-SNE 2D visualization for Future-Shot* (b) *t-SNE 2D visualization for Softmax*

Figure 5.7: Clustering and effect of point mutations. **(a)** Each circle represents a DNA sequence, with its color highlighting its lab-of-origin. Each plus sign represents a lab. They are projected from 200D to 2D using t-SNE for presentation purposes. **(b)** t-SNE visualization of the 3072D last hidden layer of the softmax model.

Then a deeper analysis of both models' predictions was performed, where it was possible to see similarities and differences between the results presented. In general, the future-shot model and the softmax model present an intersection of 94% in the results,

in which 16,481 plasmids of 18,816 have the lab-of-origin correctly positioned in the top-10. On the other hand, the two models mispredicted 1,222 plasmids. The future-shot model correctly ranks 637 plasmids that the softmax model could not, and the opposite happens for 476 samples. Figure 5.8 presents the t-SNE of the test set plasmids labeled when they were correctly or wrongly predicted by both models. It is easy to see that most mispredicted plasmids are placed in the center of the figure, where it does not have well-defined clusters. It is also in that region where more samples are correctly predicted by future-shot only, indicating that this method works better for plasmids with no prominent features.

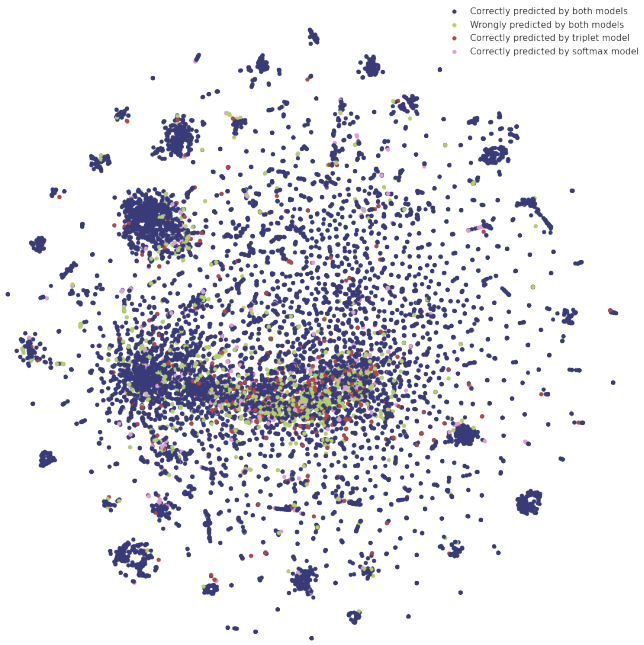


Figure 5.8: t-SNE 2D visualization of predictions. Each point (plasmid) is labeled as one of the following: correctly predicted by both models, wrongly predicted by both models, correctly predicted by the future-shot model, or correctly predicted by the softmax model. The future-shot model's embeddings are used for visualization as they create better-defined clusters.

It is also essential to understand the robustness of the model, as small changes in plasmids can frequently occur. For that reason, random point mutations were performed in the sequence from a lab and the ranking of the correct lab generated by the model was reported. Figure 5.9 shows the mean and median of correct positions after ten runs for each number of point mutations (from 0 to 1000 points). It was found that virtually all runs with up to 1000 mutations predicted the correct lab within the top 10 guesses. By increasing the number of mutations, the average runs become unstable, with the average position being higher for all cases above 400 mutations. However, if one examines the median, even with 800 mutations, the median rank for the correct lab remains within

the top 10. It indicates that the proposed model is robust to most sequence perturbations, excluding cases where these mutations affect essential features for the model's prediction.

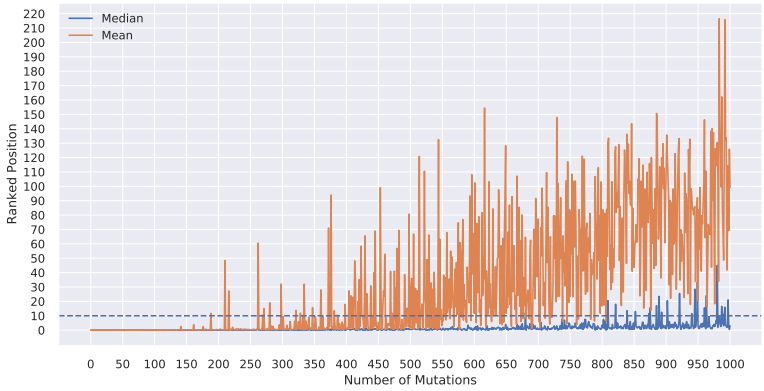


Figure 5.9: Mean and median of 100 runs of the position of the correct lab in the model prediction ranking is shown, with the number of mutations ranging from 1 to 1000.

All sequences were analyzed to discover the importance of each plasmid feature to the model output. Unlike perturbation analysis in each sequence, it was used more recent methods that generated better insights into the interpretation of the model. The method is based on integrated gradients [109]. The idea is to compute the gradient of the model's output relative to the embedding token layer. It makes it possible to visualize the importance of each token for the model's prediction. After calculating the integrated gradient for all sequences, one obtains the token importance of each lab by averaging all sequences in that lab. The same process can be performed with all sequences to get the most seen tokens in the dataset.

As can seen from Figure 5.10, some tokens appear to be shared by all labs. When generating the token importance of a lab, one can subtract it from the most seen tokens in the dataset to obtain relative importance. It allows one to examine those tokens and those not to be expected for a particular lab. Furthermore, the token importance of one lab can be compared with the most distant lab in the embedded space. The graphs of token importance from the two labs are essentially mirrored, indicating that tokens are pretty different in each case. Figures 5.10, 5.11, 5.12, and 5.13 shows these analyses, where Normalized Token Importance (NTI) is plotted as a function of the token in the left-hand column. The sequences with the most significant token importance are highlighted in the right-hand column.

This work further explored this model by looking at the token importance for David Root's lab (Figure 5.11). This analysis shows a typical cluster of sequences from this lab, allowing us to identify the potential design signatures. Furthermore, this lab is the furthest lab from the "unknown engineered" category. This class is a mixture of possible

labs and has poorly defined features. The fact that David Root's lab is the most different from this class suggests it has well-defined and perhaps highly unique design choices. Figure 5.12 shows the difference between this lab and all labs. It can be noted that for the "unknown engineered" class the scale for the normalized token importance is shallow, and the color gradient is mostly red (see Figure 5.13). It demonstrates that there is not a clear design choice or discriminating feature of this category, which is to be expected as it is a mixture of many possible labs (see methods). This analysis could be repeated for any of the labs in the dataset to identify key signatures or potential collaborations based on token proximity.

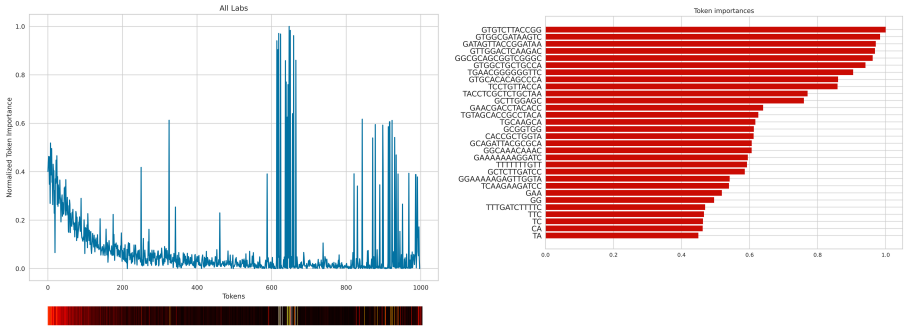


Figure 5.10: NTI for all labs obtained by averaging the token importance of all sequences and normalizing them between 0 and 1. Under the graph, a bar with a color map goes from black to white. The lighter the bar, the higher the NTI value at that point. On the right are the top 30 tokens for all data.

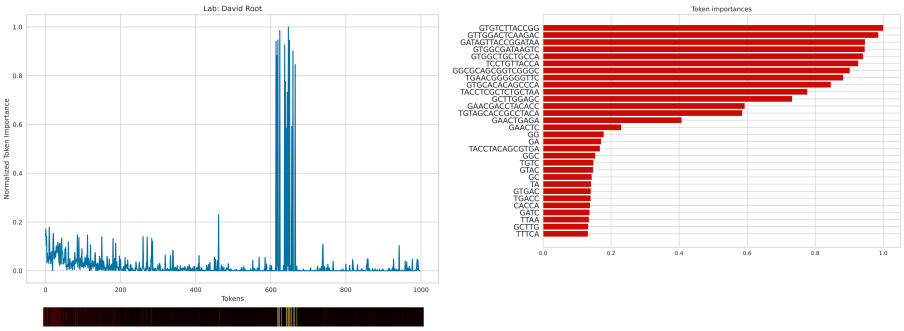


Figure 5.11: The normalized token importance for David Root's lab shows that it has similar tokens to all labs in the TOKEN ID range between 600 and 700, but in some other regions it differs a lot.

The use of integrated gradients for a single sequence is examined next. One of the primary goals of GEA approaches is to examine plasmids with unknown origins and be able to extract valuable sequence information, leading to correct assignment or further importation avenues to explore. Figure 5.14 shows that one can obtain the importance of each token within an unknown sequence with the proposed approach. When comparing the sequence token importance with the lab predicted by the model, it is visible

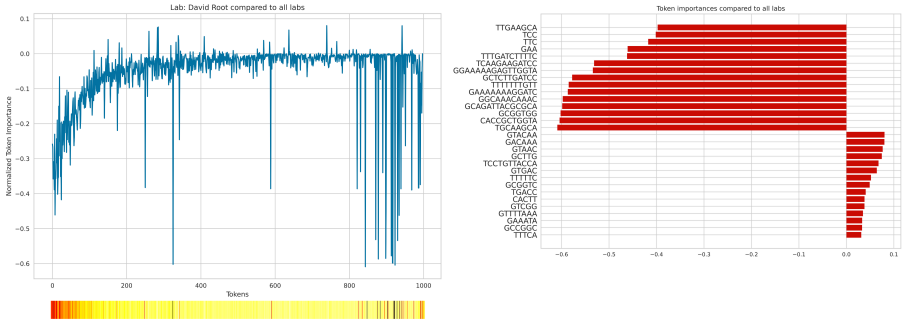


Figure 5.12: Difference between the token importance of David Root's lab and all labs. This graph highlights which tokens make a difference for this particular lab, considering the presence or absence of a token compared to other labs. On the right are the tokens that should be observed when analyzing this lab.

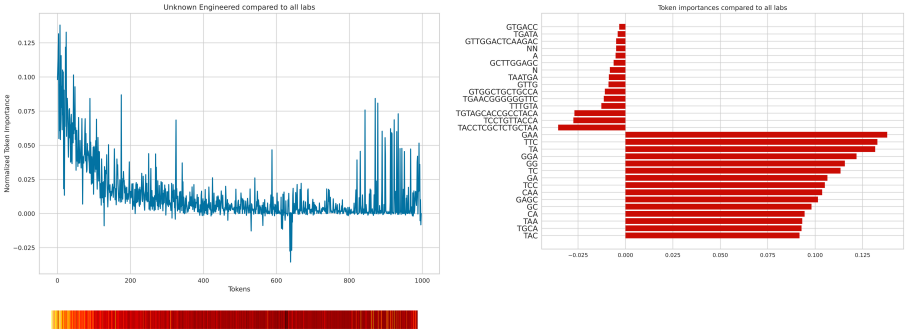


Figure 5.13: The token importance of the furthest lab to David Root's lab in the embedding space. The NTI values of the two laboratories are practically mirrored, indicating that these two laboratories have opposite characteristics.

a concordant behavior in the plots, demonstrating similarity in the highlighted features. It allows one to examine the sequence features the model uses for prediction and hence allows secondary expert evaluation of the veracity of the prediction.

Lastly, the NTI for the custom sequence designed by Alley et al was generated. [1]. This sequence combines a plasmid designed in the Omar Akbari lab (AAEL010097-Cas9, Addgene #100707 (https://www.addgene.org/100707/)) and another plasmid from Edward Boyden's Lab (pAAV-Syn-SomArchon, Addgene #126941(https://www.addgene.org/126941/)). In the analysis by Alley et al. [1], their model was uncertain how to classify this plasmid and assigned it to the category "unknown engineered". They then applied the scanning K-mer analysis to find part of the sequences that increase the probability of predicting that sequence as designed by one of the laboratories, obtaining probability peaks in parts associated with the plasmids used from each laboratory. However, their method showed a low probability of prediction for Omar Akbari lab (3%) and Boyden lab (0.3%), with superior analysis only possible when the ground-truth label is available.

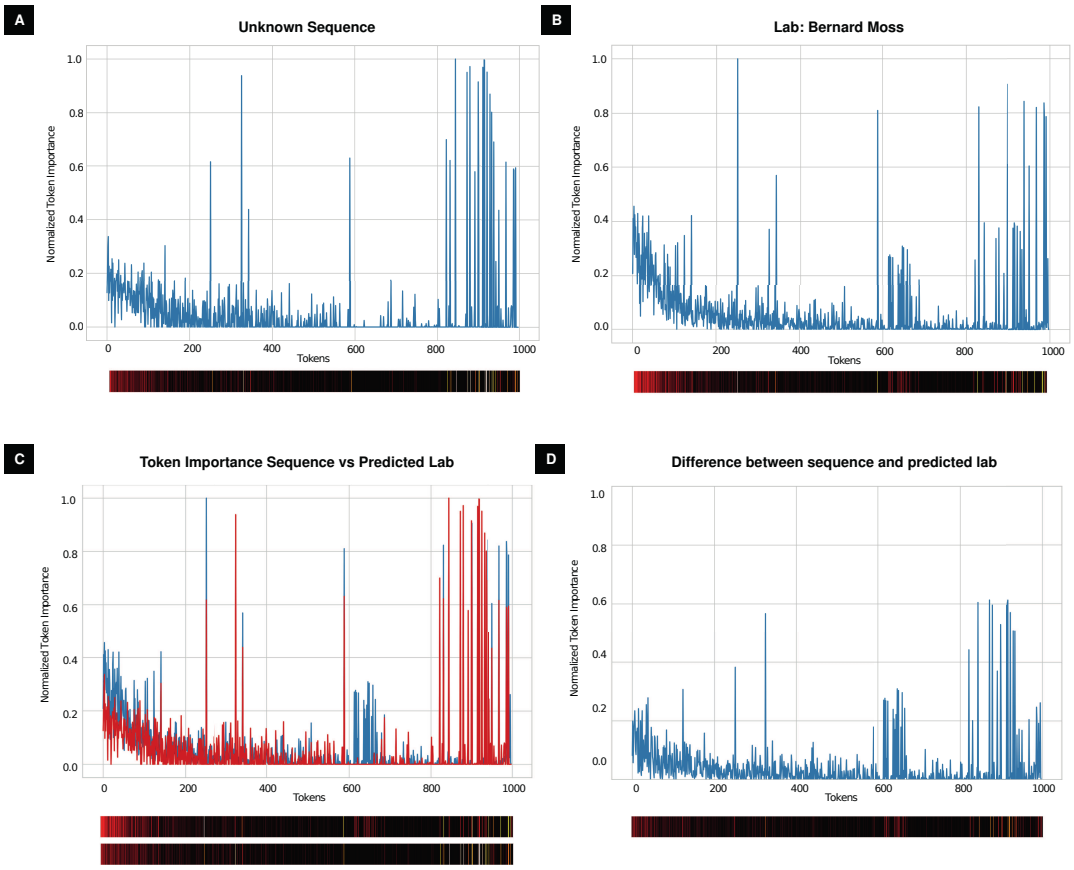


Figure 5.14: Plasmid features' importance of unknown sequences. **(a)** NTI for an unknown sequence. **(b)** The NTI for Bernard Moss's lab, which was assigned the sequence author by the future-shot model. **(c)** Comparison between tokens highlighted for the sequence and important tokens from the predicted lab. The red line represents the sequence. The first bar under the graph represents the laboratory, while the second represents the sequence. **(d)** The difference between the NTI of the sequence and the NTI of the predicted laboratory.

That same sequence was predicted using the future-shot model, which returned Akbari Lab (top1) as the closest to the sequence followed by Boyden Lab(top2), showing that this model successfully recognized the characteristics of both labs. The NTI analysis (Figure 5.15) shows that this sequence has similar token importance to the two analyzed labs. It was found that the sequence with the greatest NTI is derived from the end of the plasmid sequence located in the Ori region of the plasmid; this portion was from the Akbari lab. However, the approach of Alley et al. [1] gives a greater predicted probability of this region to the Boyden lab suggesting this helped the future-shot approach better discriminate this plasmid's lab-of-origin.

It is important to note that using the NTI interpretation method does not depend on having the ground-truth label of the sequence, being the importance defined through the weights of the already trained neural network. However, the proposed method is

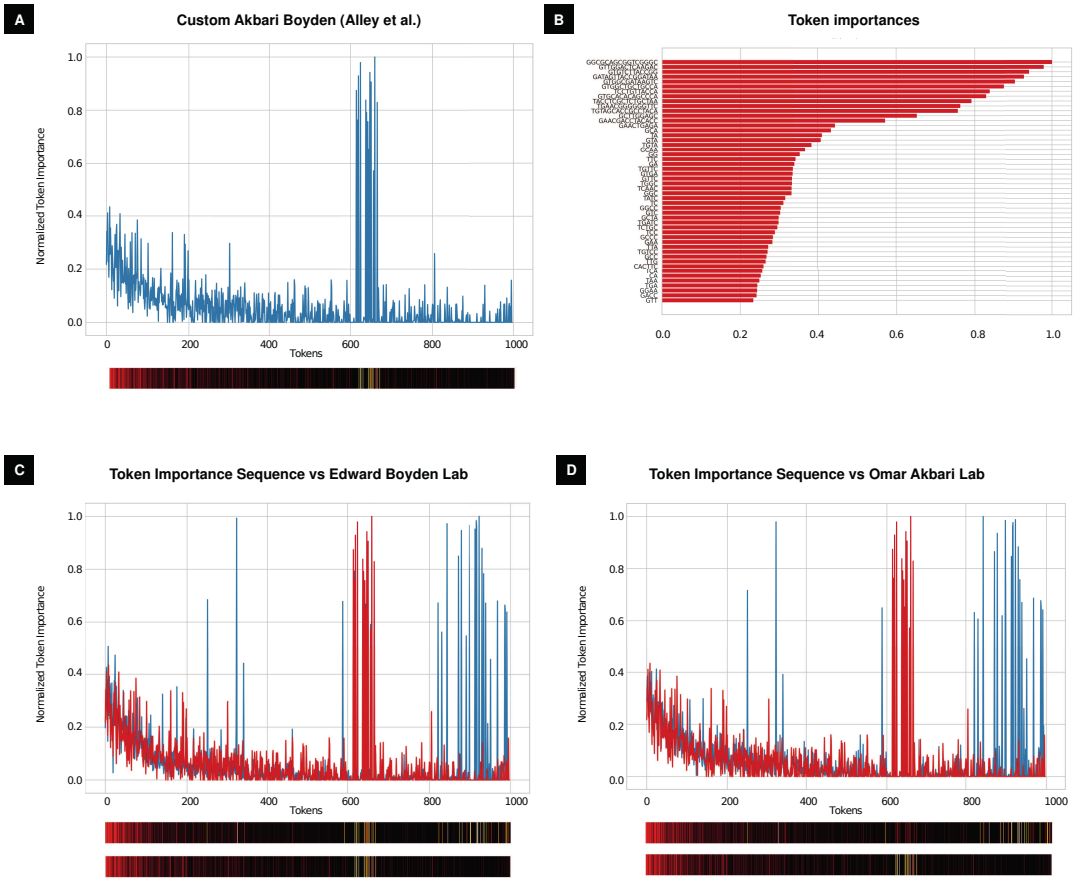


Figure 5.15: NTI for custom sequence designed by [1] and its closest two labs. **(a)** NTI for the custom sequence. **(b)** Top 30 most important tokens. **(c)** Comparison between tokens highlighted for the sequence and important tokens from the predicted Omar Akbari lab. **(d)** Comparison between tokens highlighted for the sequence and important tokens from the top2 predicted Edward Boyden lab.

directly linked to the tokens generated by the BPE since it gives importance to each grouping of characters. This grouping is optimized for natural language processing tasks but does not necessarily have a biological meaning.

5.2 Extra benchmark results

Unlike the case study, which was evaluated using top-10 accuracy, the metric for this benchmark was top-1 accuracy instead. This makes it more challenging for the model, as the correct label needs to be top-1 in terms of ranking by distance in the embedding space. Another difference is that the evaluation of this model was not as thorough as the case study. Instead, only the accuracy of both models is evaluated and then the few-shot learning evaluation is done for the future-shot model. The same process as presented in

Section 5.1.2 is followed here.

Table 5.6 shows the results of both models on the validation set and hold-out set. The hold-out set results were only computed after finding the best model using the validation set to avoid bias. As can be seen, the softmax model is slightly more accurate than the future-shot one (less than 3% better on the hold-out). Since better accuracy is not the goal of the proposed model, but instead to unlock the extra capabilities (eg. few-shot learning, clustering, and so on) instead, this small difference is not important.

Table 5.6: Accuracy on the validation set and hold-out set for both models.

Model	Val acc	Hold-out acc
Future-Shot	93.4	93.21
Softmax	**93.45**	**93.47**

Figure 5.16 shows the results of the few-shot learning experiments. Similar to what was found in the case study, it is noticeable how the accuracy improves as more data points are used to generate the embeddings. Unlike the case study, though, we see better results with less data while there are more diminishing returns as more data is used. An accuracy of 87.15% is reached using a single data point, which is a great result. Using 10% of the data points delivers 88.8% of accuracy and increasing it to 90% delivers 89.17. Those values are naturally lower than fully retraining the model (93.21%) but are sufficiently strong results to justify deploying such a model. Whenever a new class would be created, the model could start classifying it with a single example. A complete retraining of the model could later be done as more data points become available.

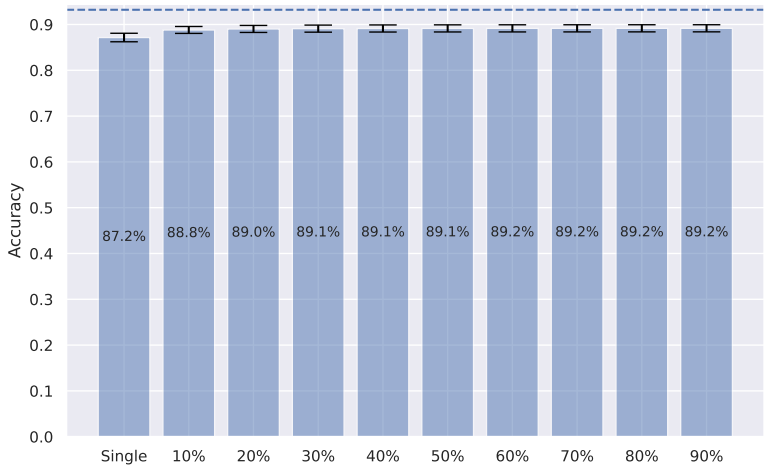


Figure 5.16: FSL Accuracy on Banking77 dataset.

Those results showcase that future-shot can be applied to other datasets in which one expects a growing number of classes. Since the method has been easily packaged into a Python library, one can easily apply it and check the results.

5.3 Final considerations

Metric learning is quite common in other areas, such as recommendation systems [42, 130], but has not been applied to genetic engineering attribution before. This methodology improved the accuracy compared to a softmax model, reaching 90.4% top-10 accuracy, a 5.4% improvement compared to the former state-of-the-art. Furthermore, it has several advantages, such as creating vector representations of labs, comparing and clustering DNA sequences and labs-of-origin, and examining design style and robustness to unseen labs.

Another advantage is dealing with a sequence from an unknown lab. A regular classifier model does not usually know how to handle such uncertainty. Typically, it spreads probabilities for each lab, always summing up to 1.0. Hence, it always assigns any plasmid sequence to one of the known labs. Meanwhile, one can check the distance from this sequence to the known labs' embeddings and decide if it is from an unknown one if it surpasses a threshold.

This work's successful use of the metric learning approach can also support future applications in other DNA data-based computational problems. Furthermore, the few-shot learning results allow the possibility of identifying new laboratories with few samples and even just a single genetic sequence. There is a trade-off between sample quantity and model accuracy, but the author believes such a methodology could be helpful in extreme cases. Finally, these embeddings are feature-rich, which means one can use them as input for other machine learning models, tackling other problems. For example, one can extract the defining signatures for labs and compare them to others using this approach.

Although it was presented a state-of-the-art classifier and a training methodology with interpretability, the approach has shortcomings. Accuracy has room for improvement, and the author believes future work should delve into more modern NLP approaches. It could include using more advanced machine learning architectures, other pre-processing methods, and data augmentation techniques leading to better convergence and algorithm training. One future work direction would be to adapt the model for diploid features [30]. The author also believes that techniques like Transformers [116] and Graph Convolutional Networks [92] would be good candidates for this task since patterns inside the sequence can be considered contextualized, for which Transformers generally show good performance [116].

Deep learning approaches rely on large amounts of high-quality data. The FSL approach was designed to handle laboratories with few samples, generally classified as "unknown engineered". However, neither the data nor the evaluation methods for GEA are concise between different works. To better track advances in the area, future work

from the biotechnology community should focus on creating more robust datasets with a specific test set for evaluation, similar to what can be seen in computer vision [61, 19, 4, 53] and natural language processing [84, 117, 128, 131] tasks. It will increase the reliability of the results and prevent accidental misreporting.

The interpretation method using integrated gradients highlights essential tokens within the sequence in the model view. It was shown that the method presented does not depend on the ground-truth label, needs only one execution for the entire sequence, and can help specialists in further investigations. However, the tokens generated by BPE are purely analyzed from the perspective of pattern recognition in natural language processing, and they may not have biological representation. The author believes that a significant improvement would be to map subsequences with biological characteristics into tokens, similar to the process performed by BPE but done by a genetic engineering specialist. Future work focusing on this will help improve interpretation techniques and the understanding of essential patterns by computer scientists who are not experts in genetic engineering. The author hopes that the methodology and results presented here stimulate new directions for future research.

Lastly, the methodology developed here was also applied to another benchmark dataset (Banking77) and also showed promising results, proving that this method can be applied to other use cases successfully. To make it easier to do, this work also includes an easy-to-use library that can be used by other researchers to tackle similar problems.

CHAPTER 6

Conclusion and future work

This thesis has introduced Future-Shot, a pioneering approach for high-dimensional multiclass classification in dynamic environments. By leveraging metric learning techniques, Future-Shot achieves remarkable accuracy in lab-of-origin prediction tasks, outperforming existing methods while not requiring retraining to accommodate new classes. Beyond its success in the lab-of-origin prediction domain, Future-Shot can be applied to other classification tasks, as demonstrated by the extra benchmark conducted. For that matter, an open-source library has been developed and made available to allow further research in other areas. This thesis shows all the utility that the approach brings to the table, like allowing the use of FSL to deal with new classes, being able to do clustering, and even allowing for the reuse of the learned embeddings for other models.

Future research avenues abound. Extending this work involves exploring the method's scalability and limits, such as the threshold for adding new classes without retraining and the overall class capacity. Furthermore, applying Future-Shot to various domains and modalities will validate its broad applicability. Investigating advanced metric learning techniques like Multi-Class N-pair Loss [105] offers avenues for enhancing convergence and speed. Additionally, refining the lab-of-origin prediction task through the exploration of more sophisticated models (like transformers) promises further advancements.

References

[1] Ethan C. Alley et al. “A machine learning toolkit for genetic engineering attribution to facilitate biosecurity”. In: *Nature Communications* 11.1 (Dec. 2020), p. 6293. ISSN: 2041-1723. DOI: 10.1038/s41467-020-19612-0. URL: https://doi.org/10.1038/s41467-020-19612-0.

[2] Ethan C. Alley et al. “Attribution of genetic engineering: A practical and accurate machine-learning toolkit for biosecurity”. In: *bioRxiv* (2020). DOI: 10.1101/2020.08.22.262576. eprint: https://www.biorxiv.org/content/early/2020/08/22/2020.08.22.262576.full.pdf. URL: https://www.biorxiv.org/content/early/2020/08/22/2020.08.22.262576.

[3] Stephen F. Altschul et al. “Basic local alignment search tool”. In: *Journal of Molecular Biology* 215.3 (Oct. 1990), pp. 403–410. DOI: 10.1016/s0022-2836(05)80360-2. URL: https://doi.org/10.1016/s0022-2836(05)80360-2.

[4] Mykhaylo Andriluka et al. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471.

[5] F. Anwaar et al. “Hrs-ce: a hybrid framework to integrate content embeddings in recommender systems for cold start items”. In: *Journal of Computational Science* 29 (2018), pp. 9–18. DOI: 10.1016/j.jocs.2018.09.008.

[6] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5 2 (1994), pp. 157–66.

[7] Bonnie Berger, Michael Waterman, and Yun Yu. “Levenshtein Distance, Sequence Comparison and Biological Database Search”. In: *IEEE Transactions on Information Theory* PP (May 2020), pp. 1–1. DOI: 10.1109/TIT.2020.2996543.

[8] T. Boulmaiz, Mawloud Guermoui, and Hamouda Boutaghane. “Impact of training data size on the LSTM performances for rainfall–runoff modeling”. In: *Modeling Earth Systems and Environment* (2020), pp. 1–12.

[9] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *ArXiv* abs/2005.14165 (2020).

[10] Iñigo Casanueva et al. “Efficient Intent Detection with Dual Sentence Encoders”. In: *Proceedings of the 2nd Workshop on NLP for ConvAI - ACL 2020*. Data available at https://github.com/PolyAI-LDN/task-specific-datasets. Mar. 2020. URL: https://arxiv.org/abs/2003.04807.

[11] Supriyo Chakraborty et al. “Interpretability of deep learning models: A survey of results”. In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (2017), pp. 1–6.

[12] Davide Chicco. “Ten quick tips for machine learning in computational biology”. In: *BioData Mining* 10.1 (Dec. 2017). DOI: 10.1186/s13040-017-0155-3. URL: https://doi.org/10.1186/s13040-017-0155-3.

[13] Allen Chieng Hoon Choong and Nung Kion Lee. “Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method”. In: *2017 International Conference on Computer and Drone Applications (IConDA)*. IEEE, Nov. 2017. DOI: 10.1109/iconda.2017.8270400. URL: https://doi.org/10.1109/iconda.2017.8270400.

[14] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)* 1 (2005), 539–546 vol. 1.

[15] Anna Choromanska, Alekh Agarwal, and John Langford. “Extreme multi class classification”. In: *NIPS Workshop: eXtreme Classification*. Submitted. 2013.

[16] Nicole Dalia Cilia et al. “What is the minimum training data size to reliably identify writers in medieval manuscripts?” In: *Pattern Recognit. Lett.* 129 (2020), pp. 198–204.

[17] Thomas M. Cover and Peter E. Hart. “Nearest neighbor pattern classification”. In: *IEEE Trans. Inf. Theory* 13 (1967), pp. 21–27.

[18] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[19] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[20] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. “ArcFace: Additive Angular Margin Loss for Deep Face Recognition”. In: *CoRR* abs/1801.07698 (2018). arXiv: 1801.07698. URL: http://arxiv.org/abs/1801.07698.

[21] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL*. 2019.

[22] Finale Doshi-Velez and Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. arXiv: 1702.08608 [stat.ML].

[23] Kirolos Eskandar. “Revolutionizing biotechnology and bioengineering: unleashing the power of innovation”. In: *Journal of Applied Biotechnology & Bioengineering* (2023). URL: https://api.semanticscholar.org/CorpusID:259883441.

[24] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://www.pytorchlightning.ai.

[25] Li Fei-Fei, R. Fergus, and P. Perona. “One-shot learning of object categories”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.4 (Apr. 2006), pp. 594–611. DOI: 10.1109/tpami.2006.79. URL: https://doi.org/10.1109/tpami.2006.79.

[26] Michael Fink. “Object Classification from a Single Example Utilizing Class Relevance Metrics”. In: *Advances in Neural Information Processing Systems*. Ed. by L. Saul, Y. Weiss, and L. Bottou. Vol. 17. MIT Press, 2005. URL: https://proceedings.neurips.cc/paper/2004/file/ef1e491a766ce3127556063d49bc2f98-Paper.pdf.

[27] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *CoRR* abs/1703.03400 (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[28] Philip Gage. “A New Algorithm for Data Compression”. In: *C Users J.* 12.2 (Feb. 1994), pp. 23–38. ISSN: 0898-9788.

[29] Tianyu Gao, Adam Fisch, and Danqi Chen. *Making Pre-trained Language Models Better Few-shot Learners*. 2021. arXiv: 2012.15723 [cs.CL].

[30] Shilpa Garg. “Computational methods for chromosome-scale haplotype reconstruction”. In: *Genome Biology* 22.1 (Apr. 2021). DOI: 10.1186/s13059-021-02328-9. URL: https://doi.org/10.1186/s13059-021-02328-9.

[31] Federico Girosi, Michael Jones, and Tomaso Poggio. “Regularization Theory and Neural Networks Architectures”. In: *Neural Computation* 7.2 (Mar. 1995), pp. 219–269. ISSN: 0899-7667. DOI: 10.1162/neco.1995.7.2.219. eprint: https://direct.mit.edu/neco/article-pdf/7/2/219/812917/neco.1995.7.2.219.pdf. URL: https://doi.org/10.1162/neco.1995.7.2.219.

[32] John Hale et al. “Text Genre and Training Data Size in Human-like Parsing”. In: *EMNLP*. 2019.

[33] Xu Han et al. *PTR: Prompt Tuning with Rules for Text Classification*. 2021. arXiv: 2105.11259 [cs.CL].

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. Chap. 7. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[35] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[36] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. 2017. arXiv: 1703.07737 [cs.CV].

[37] Hideitsu Hino. *Active Learning: Problem Settings and Recent Developments*. 2020. arXiv: 2012.04225 [cs.LG].

[38] Geoffrey Hinton and Sam Roweis. “Stochastic Neighbor Embedding”. In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS’02. Cambridge, MA, USA: MIT Press, 2002, pp. 857–864.

[39] Elad Hoffer and Nir Ailon. *Deep metric learning using Triplet network*. 2018. arXiv: 1412.6622 [cs.LG].

[40] John J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” In: *Proceedings of the National Academy of Sciences of the United States of America* 79 8 (1982), pp. 2554–8.

[41] Matin Hosseinzadeh et al. “Deep learning–assisted prostate cancer detection on bi-parametric MRI: minimum training data size requirements and effect of prior knowledge”. In: *European Radiology* 32 (2021), pp. 2224–2234.

[42] Cheng-Kang Hsieh et al. “Collaborative Metric Learning”. In: International World Wide Web Conferences Steering Committee, Apr. 2017. DOI: 10.1145/3038912.3052639. URL: https://doi.org/10.1145/3038912.3052639.

[43] Shell Xu Hu et al. *Pushing the Limits of Simple Pipelines for Few-Shot Learning: External Data and Fine-Tuning Make a Difference*. 2022. arXiv: 2204.07305 [cs.CV].

[44] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[45] Joanne Kamens. “The Addgene repository: an international nonprofit plasmid and data resource”. In: 43.D1 (Nov. 2014), pp. D1152–D1157. DOI: 10.1093/nar/gku893. URL: https://doi.org/10.1093/nar/gku893.

[46] Md Rezaul Karim et al. “Deep learning-based clustering approaches for bioinformatics”. In: 22.1 (Feb. 2020), pp. 393–415. DOI: 10.1093/bib/bbz170. URL: https://doi.org/10.1093/bib/bbz170.

[47] Shachar Kaufman, Saharon Rosset, and Claudia Perlich. “Leakage in Data Mining: Formulation, Detection, and Avoidance”. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’11. San Diego, California, USA: Association for Computing Machinery, 2011, pp. 556–563. ISBN: 9781450308137. DOI: 10.1145/2020408.2020496. URL: https://doi.org/10.1145/2020408.2020496.

[48] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*. 2014. arXiv: 1408.5882 [cs.CL].

[49] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[50] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *CoRR* abs/1706.02515 (2017). arXiv: 1706.02515. URL: http://arxiv.org/abs/1706.02515.

[51] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. “Siamese Neural Networks for One-shot Image Recognition”. In: 2015.

[52] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. *Learning Active Learning from Data*. 2017. arXiv: 1703.03365 [cs.LG].

[53] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[55] Brian Kulis. “Metric Learning: A Survey”. In: *Foundations and Trends® in Machine Learning* 5.4 (2013), pp. 287–364. ISSN: 1935-8237. DOI: 10.1561/2200000019. URL: http://dx.doi.org/10.1561/2200000019.

[56] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proc. IEEE* 86 (1998), pp. 2278–2324.

[57] J. Lee et al. “Deep content-user embedding model for music recommendation”. In: (2018). DOI: 10.48550/arxiv.1807.06786.

[58] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. arXiv: 2104.08691 [cs.CL].

[59] Quentin Lhoest et al. “Datasets: A Community Library for Natural Language Processing”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. arXiv: 2109.02846 [cs.CL]. URL: https://aclanthology.org/2021.emnlp-demo.21.

[60] Haoting Liang et al. “Personalized Music Recommendation with Triplet Network”. In: (July 2019).

[61] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list. 2014. URL: http://arxiv.org/abs/1405.0312.

[62] Zachary Chase Lipton. “A Critical Review of Recurrent Neural Networks for Sequence Learning”. In: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[63] C. Liu et al. “learning a few-shot embedding model with contrastive learning”. In: *Proceedings of the Aaai Conference on Artificial Intelligence* 35 (10 2021), pp. 8635–8643. DOI: 10.1609/aaai.v35i10.17047.

[64] Weiyang Liu et al. “SphereFace: Deep Hypersphere Embedding for Face Recognition”. In: *CoRR* abs/1704.08063 (2017). arXiv: 1704.08063. URL: http://arxiv.org/abs/1704.08063.

[65] Zhibin Lv et al. “A Convolutional Neural Network Using Dinucleotide One-hot Encoder for identifying DNA N6-Methyladenine Sites in the Rice Genome”. In: *Neurocomputing* 422 (Jan. 2021), pp. 214–221. DOI: 10.1016/j.neucom.2020.09.056. URL: https://doi.org/10.1016/j.neucom.2020.09.056.

[66] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[67] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[68] Agnieszka Mikolajczyk and Michal Grochowski. “Data augmentation for improving deep learning in image classification problem”. In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. IEEE, May 2018. DOI: 10.1109/iiphdw.2018.8388338. URL: https://doi.org/10.1109/iiphdw.2018.8388338.

[69] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[70] Nikita Moshkov et al. “Test-time augmentation for deep learning-based cell segmentation on microscopy images”. In: *Scientific Reports* 10.1 (Mar. 2020). DOI: 10.1038/s41598-020-61808-3. URL: https://doi.org/10.1038/s41598-020-61808-3.

[71] I. Muniz, F. H. F. Camargo, and A. Marques. *Ranking labs-of-origin for genetically engineered DNA using Metric Learning*. 2021. DOI: 10.48550/ARXIV.2107.07878. URL: https://arxiv.org/abs/2107.07878.

[72] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. *PyTorch Metric Learning*. 2020. arXiv: 2008.09164 [cs.CV].

[73] Allen Newell. “Perceptrons. An Introduction to Computational Geometry. Marvin Minsky and Seymour Papert. M.I.T. Press, Cambridge, Mass., 1969. vi + 258 pp., illus. Cloth, $12; paper, $4.95”. In: *Science* 165 (1969), pp. 780–782.

[74] Chenri Ni et al. *On the Calibration of Multiclass Classification with Rejection*. 2019. arXiv: 1901.10655 [stat.ML].

[75] Alex Nichol, Joshua Achiam, and John Schulman. “On First-Order Meta-Learning Algorithms”. In: *CoRR* abs/1803.02999 (2018). arXiv: 1803.02999. URL: http://arxiv.org/abs/1803.02999.

[76] Alec A. K. Nielsen and Christopher A. Voigt. “Deep learning to predict the lab-of-origin of engineered DNA”. In: *Nature Communications* 9.1 (Aug. 2018), p. 3135. ISSN: 2041-1723. DOI: 10.1038/s41467-018-05378-z. URL: https://doi.org/10.1038/s41467-018-05378-z.

[77] Mahamed Omran, Andries Engelbrecht, and Ayed Salman. “An overview of clustering methods”. In: *Intell. Data Anal.* 11 (Nov. 2007), pp. 583–605. DOI: 10.3233/IDA-2007-11602.

[78] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *CoRR* abs/1807.03748 (2018). arXiv: 1807.03748. URL: http://arxiv.org/abs/1807.03748.

[79] Boris N. Oreshkin, Pau Rodrıguez López, and Alexandre Lacoste. “TADAM: Task dependent adaptive metric for improved few-shot learning”. In: *CoRR* abs/1805.10123 (2018). arXiv: 1805.10123. URL: http://arxiv.org/abs/1805.10123.

[80] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[81] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[82] Roberto Polli, Erik Wilde, and Eemeli Aro. *YAML Media Type*. Internet-Draft draft-ietf-httpapi-yaml-mediatypes-10. Work in Progress. Internet Engineering Task Force, Aug. 2023. 17 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-httpapi-yaml-mediatypes/10/.

[83] Guanghui Qin and Jason Eisner. *Learning How to Ask: Querying LMs with Mixtures of Soft Prompts*. 2021. arXiv: 2104.06599 [cs.CL].

[84] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL].

[85] Ankit Singh Rawat et al. “Sampled Softmax with Random Fourier Features”. In: *CoRR* abs/1907.10747 (2019). arXiv: 1907.10747. URL: http://arxiv.org/abs/1907.10747.

[86] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: http://arxiv.org/abs/1908.10084.

[87] Laria Reynolds and Kyle McDonell. *Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm*. 2021. arXiv: 2102.07350 [cs.CL].

[88] F. Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957.

[89] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[90] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[91] Andrei A. Rusu et al. “Meta-Learning with Latent Embedding Optimization”. In: *CoRR* abs/1807.05960 (2018). arXiv: 1807.05960. URL: http://arxiv.org/abs/1807.05960.

[92] Franco Scarselli et al. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[93] J. Ben Schafer et al. “Collaborative Filtering Recommender Systems”. In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 291–324. ISBN: 9783540720782.

[94] Timo Schick, Helmut Schmid, and Hinrich Schütze. *Automatically Identifying Words That Can Serve as Labels for Few-Shot Text Classification*. 2020. arXiv: 2010.13641 [cs.CL].

[95] Timo Schick and Hinrich Schütze. *Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference*. 2021. arXiv: 2001.07676 [cs.CL].

[96] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). DOI: 10.1109/cvpr.2015.7298682. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[97] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”. In: *CoRR* abs/1808.03314 (2018). arXiv: 1808.03314. URL: http://arxiv.org/abs/1808.03314.

[98] Taylor Shin et al. *AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts*. 2020. arXiv: 2010.15980 [cs.CL].

[99] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6 (2019), pp. 1–48.

[100] Connor Shorten, Taghi M. Khoshgoftaar, and Borko Furht. “Text Data Augmentation for Deep Learning”. In: *Journal of Big Data* 8 (2021).

[101] Leslie N. Smith and Nicholay Topin. “Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates”. In: *CoRR* abs/1708.07120 (2017). arXiv: 1708.07120. URL: http://arxiv.org/abs/1708.07120.

[102] Jake Snell, Kevin Swersky, and Richard S. Zemel. “Prototypical Networks for Few-shot Learning”. In: *CoRR* abs/1703.05175 (2017). arXiv: 1703.05175. URL: http://arxiv.org/abs/1703.05175.

[103] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian optimization of machine learning algorithms”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 2951–2959.

[104] Igor M. Soares et al. “Improving lab-of-origin prediction of genetically engineered plasmids via deep metric learning”. In: *Nature Computational Science* 2.4 (Apr. 2022), pp. 253–264. ISSN: 2662-8457. DOI: 10.1038/s43588-022-00234-z. URL: https://doi.org/10.1038/s43588-022-00234-z.

[105] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: *NIPS*. 2016.

[106] Hyun Oh Song et al. “Deep Metric Learning via Lifted Structured Feature Embedding”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 4004–4012.

[107] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[108] Xiaofei Sun et al. *Text Classification via Large Language Models*. 2023. arXiv: 2305.08377 [cs.CL].

[109] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *CoRR* abs/1703.01365 (2017). arXiv: 1703.01365. URL: http://arxiv.org/abs/1703.01365.

[110] Flood Sung et al. “Learning to Compare: Relation Network for Few-Shot Learning”. In: *CoRR* abs/1711.06025 (2017). arXiv: 1711.06025. URL: http://arxiv.org/abs/1711.06025.

[111] Robert L. Thorndike. “Who belongs in the family?” In: *Psychometrika* 18.4 (Dec. 1953), pp. 267–276. ISSN: 1860-0980. DOI: 10.1007/BF02289263. URL: https://doi.org/10.1007/BF02289263.

[112] Christos Thrampoulidis, Samet Oymak, and Mahdi Soltanolkotabi. “Theoretical Insights Into Multiclass Classification: A High-dimensional Asymptotic View”. In: *ArXiv* abs/2011.07729 (2020).

[113] Rafael Torres et al. “Training Data Size Requirements for Topic Classification in a Speech-Oriented Guidance System”. In: 2010.

[114] Viet-Anh Tran et al. “Improving Collaborative Metric Learning with Efficient Negative Sampling”. In: *CoRR* abs/1909.10912 (2019). arXiv: 1909.10912. URL: http://arxiv.org/abs/1909.10912.

[115] “Using metric learning to identify the lab-of-origin of engineered DNA”. In: *Nature Computational Science* 2.5 (May 2022), pp. 296–297. ISSN: 2662-8457. DOI: 10.1038/s43588-022-00240-1. URL: https://doi.org/10.1038/s43588-022-00240-1.

[116] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[117] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. cite arxiv:1804.07461Comment: https://gluebenchmark.com/. 2018. URL: http://arxiv.org/abs/1804.07461.

[118] Feng Wang et al. “Additive Margin Softmax for Face Verification”. In: *CoRR* abs/1801.05599 (2018). arXiv: 1801.05599. URL: http://arxiv.org/abs/1801.05599.

[119] Feng Wang et al. “NormFace: L_2 Hypersphere Embedding for Face Verification”. In: *CoRR* abs/1704.06369 (2017). arXiv: 1704.06369. URL: http://arxiv.org/abs/1704.06369.

[120] Hao Wang et al. “CosFace: Large Margin Cosine Loss for Deep Face Recognition”. In: *CoRR* abs/1801.09414 (2018). arXiv: 1801.09414. URL: http://arxiv.org/abs/1801.09414.

[121] Qi Wang et al. “Faster pan-genome construction for efficient differentiation of naturally occurring and engineered plasmids with plaster”. In: *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019.

[122] Qi Wang et al. “PlasmidHawk improves lab of origin prediction of engineered plasmids using sequence alignment”. In: *Nature Communications* 12.1 (Feb. 2021), p. 1167. ISSN: 2041-1723. DOI: 10.1038/s41467-021-21180-w. URL: https://doi.org/10.1038/s41467-021-21180-w.

[123] Xun Wang et al. “Cross-Batch Memory for Embedding Learning”. In: *CoRR* abs/1912.06798 (2019). arXiv: 1912.06798. URL: http://arxiv.org/abs/1912.06798.

[124] Yaqing Wang et al. “Generalizing from a Few Examples”. In: *ACM Computing Surveys* 53.3 (July 2020), pp. 1–34. DOI: 10.1145/3386252. URL: https://doi.org/10.1145/3386252.

[125] Yaqing Wang et al. “Generalizing from a Few Examples: A Survey on Few-Shot Learning”. In: *arXiv: Learning* (2019).

[126] Jules White et al. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. 2023. arXiv: 2302.11382 [cs.SE].

[127] Bernard Widrow and Michael A. Lehr. “30 years of adaptive neural networks: perceptron, Madaline, and backpropagation”. In: *Proc. IEEE* 78 (1990), pp. 1415–1442.

[128] Adina Williams, Nikita Nangia, and Samuel Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: http://aclweb.org/anthology/N18-1101.

[129] Dongkuan Xu and Yingjie Tian. “A Comprehensive Survey of Clustering Algorithms”. In: 2.2 (June 2015), pp. 165–193. DOI: 10.1007/s40745-015-0040-1. URL: https://doi.org/10.1007/s40745-015-0040-1.

[130] Junliang Yu et al. “A Social Recommender Based on Factorization and Distance Metric Learning”. In: 5 (2017), pp. 21557–21566. DOI: 10.1109/access.2017.2762459. URL: https://doi.org/10.1109/access.2017.2762459.

[131] Rowan Zellers et al. *SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference*. 2018. arXiv: 1808.05326 [cs.CL].

[132] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. “Hierarchical Agglomerative Clustering”. In: *Encyclopedia of Systems Biology*. Ed. by Werner Dubitzky et al. New York, NY: Springer New York, 2013, pp. 886–887. ISBN: 978-1-4419-9863-7. DOI: 10.1007/978-1-4419-9863-7_1371. URL: https://doi.org/10.1007/978-1-4419-9863-7_1371.

[133] Wanwan Zheng and Mingzhe Jin. “Effects of Training Data Size and Class Imbalance on the Performance of Classifiers”. In: *Communications in Computer and Information Science* (2019).

[134] Quan Zou et al. “Gene2vec: gene subsequence embedding for prediction of mammalian N6-methyladenosine sites from mRNA”. In: *RNA* 25.2 (Nov. 2018), pp. 205–218. DOI: 10.1261/rna.069112.118. URL: https://doi.org/10.1261/rna.069112.118.