Fall 12-17-2021

Nitrogenase Iron Protein Classification using CNN Neural Network

Amer Rez
San Jose State University

Nitrogenase Iron Protein Classification using CNN Neural Network

A Project Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by Amer Rez

November 2021

The Designated Project Committee Approves the Project Titled

Nitrogenase Iron Protein Classification using CNN Neural Network

By

Amer Rez

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE SAN JOSÉ STATE UNIVERSITY

November 2021

Dr. Nada Attar	Department of Computer Science
Dr. Philip Heller	Department of Computer Science
Dr. Bilal Alsallakh	Machine Learning & Visualization Scientist

Abstract

The nitrogenase iron protein (NifH) is extensively used to study nitrogen fixation, the ecologically vital process of reducing atmospheric nitrogen to a bioavailable form. The discovery rate of novel NifH sequences is high, and there is an ongoing need for software tools to mine NifH records from the GenBank repository. Since record annotations are unreliable, because they contain errors, classifiers based on sequence alone are required. The ARBitrator classifier is highly successful but must be initialized by extensive manual effort. A Deep Learning approach could substantially reduce manual intervention. However, attempts to build a character-based Deep Learning NifH classifier were unsuccessful. We hypothesized that we could generate visual representations of protein sequences and use a Convolutional Neural Network to classify the representations. Here we present the resulting classifier, which has achieved false positive and false negative rates of 0.19% and 0.22%, respectively.

Acknowledgements

I would like to thank God, my parents, my wife, and my family. I received tremendous support from my parents especially in my education. I would like to thank all my teachers and professors that taught me throughout my life, and professor Nada and Philip who helped me throughout this project.

Table of Contents

List of Figures

List of Tables

1. Introduction

Functional classification of nucleotide sequences is an essential component of bioinformatics analysis. In metagenomic and metatranscriptomic studies, functional classification is critical in the annotation of new genomes and curating databases of all known sequences of a particular gene of interest. In annotation projects and metagenomic and metatranscriptomic studies, community DNA is sequenced, and sequences are mapped to gene function before statistical analysis [1] [2] [3] [4]. In database curation, candidate sequences are evaluated according to a binary classifier; They are then labeled as representing or not representing the gene function of interest.

Database curation can be performed either in vitro or in silico. In in vitro curation (e.g. [5]), molecules of the gene of interest are recovered from a tissue sample using degenerate PCR primers and are then sequenced. While highly accurate, this approach is slow and expensive. In in silico curation, a software binary classifier analyzes records from a large annotated general-purpose sequence database, typically GenBank [6], retaining records determined to represent the gene function of interest. Classification can consider annotations only, nucleotide or protein sequence only, or some combination. Annotations are not consistently reliable [7], so annotation-only classifiers should be used with caution.

The nifH gene, which encodes two identical subunits of the iron protein of the nitrogenase enzyme, is the subject of several publicly available software-curated databases. nifH is the most conserved, and hence most studied, of the three components of the nitrogenase enzyme, which reduces inert atmospheric nitrogen (N2) into bioavailable ammonia [8], a necessary precursor to the biosynthesis of nucleic acids and proteins. Nitrogen-reducing organisms, or diazotrophs, are sparsely distributed within the prokaryotic tree of life, primarily in the phylum Cyanobacteria, a photosynthesizing bacteria [18]. However, they also exist in other unicellular phyla [9]. Therefore, curating a database of nifH sequences requires classification of all known prokaryotic sequences, not of any particular clade. Four approaches have been used to generate nifH databases. All of which have mined the GenBank nr or nt databases and then applied customized classifiers. The Global Census of Nitrogenase Diversity (2011) analyzed annotations [10]. The FunGene database (2013) used a Hidden Markov Model [11]. In 2014, the Global Census of Nitrogenase Diversity authors published a database that classified based on alignment, G+C content, and sequence length [12]. Also, in 2014, Heller, working in the Zehr research group at The University of Califonia Santa Cruz, published the ARBitrator database [13], which classifies based on a blastp search of GenBank's nr database followed by a reverse PSI-blast [14] against GenBank's Conserved Domain Database [15].

The ARBitrator classifier is highly accurate; its false-positive rate was estimated at 0.033%. ARBitrator's accuracy is also significantly better than FunGene and similar to the Global Census of Nitrogenase Diversity [13]. When the classifier was adapted for the cytochrome c oxidase subunit I (COI) gene [16], accuracy was comparable to the BOLD database [5], which is curated in vitro. However, this accuracy comes at a cost. In addition to positive and negative training sets, which are required by any classifier, ARBitrator requires manual selection of a small subset of positive examples that represent sequence diversity and painstakingly determined thresholds for blastp and reverse PSI-blast hits. Here we report on our efforts to develop a nifH classifier that uses Deep Learning [20]. Since the parameters of Deep Learning classifiers are computed in silico during the training process, manual startup intervention would be minimized. Initial efforts were unpromising: when a Deep Learning neural network was trained on inputs of ASCII codes for the single-character amino acid symbols of the sequence being classified, results could not achieve accuracy greater than 93%.

We believe that the failure of the initial attempt was the result of an arbitrary choice of neural network topology. The range of options for the number of hidden layers, their sizes, and their interconnections is practically unlimited; moreover, there is little prior research on Deep Learning classification of genetic sequences to inform design choices. We therefore decided to translate the problem into a domain with an extensive body of prior literature: image processing with convolutional neural networks.

Convolutional neural networks (CNN) are extensively employed in bioinformatics [24], computer vision, speech recognition, and natural language processing [22] [23]. They have proven to achieve significant results due to efficient feature extraction [25]. They were utilized in specific bioinformatics applications like motif discovery [26], HLA class I-peptide binding prediction [27], and predicting the effects of noncoding variants [28]. Zeng et al. (2016) presented a systematic exploration of various CNN architectures for predicting DNA sequence binding using an extensive compendium of transcription factor datasets [29]. Kelly et al. (2016) introduced Basset, an open-source package to apply CNNs to learn the functional activity of DNA sequences from genomics data [30].

Research that utilized CNNs to solve non-vision-based problems gave more confidence to try this approach. They improved multiple aspects, including accuracy and performance. For example, in genetic diseases, Arena et al. (2002) used CNNs to characterize and analyze genetic data [31]. They were able to improve performance by grouping data entries to be processed together during training and inferencing. Grouping sequences allows analysis of many of them synchronously since each sequence represents a single region of an input image. Another example study used CNNs to find specific DNA sites in genomes to help modify DNA [32]. In their study, researchers built a CNN-based prediction model which outperformed previous models using mouse DNA data. Their study indicates that CNNs can be of great use for researchers in the fields of bioinformatics. A third example of utilizing CNNs in the genetics field is a study that uses a CNN model to predict protein functions by converting the protein function problem into a language translation problem [33]. The proteins in that study belong to 10 classes representing the different functions of proteins. They used a

CNN-based tool, called DeepInsight, that gave the best accuracy compared to Decision Tree, Ada-Boost, and Random Forest algorithms [38]. The fourth example is a study by Kulmanov and Hoehndorf; they took a different approach in CNN models using many CNN networks of one dimension [34]. The researchers converted the data into a 1-hot encoding representation. Each of the corresponding 1-hot sequences was used on a separate one-dimensional CNN. They experimented with several neural networks such as recurrent neural networks, long-short term memory networks, and autoencoders. CNNs gave the best results among all networks they tested.

Our research goal is to achieve a high classifier accuracy for protein sequences by converting sequences to pdf images, and using a CNN topology whose accuracy as an image classifier has previously been established. Here we describe our success with such a classifier. This report is organized as follows: Section two describes an overview of why we chose CNN as the most effective architecture for this specific problem, and introduces data description, data processing, and image sequences. Section three describes CNN background and models used in this report. Sections four to six describe the experimental model, the results, and discussion. Finally, section seven presents the conclusion of this research.

2. Overview and Data Preparation

2.1. Overview

Previous work tried scaling the images and using a 2-layer convolutional network [40]. Two max-pooling layers were used, one after each of its two convolutional layers. Images were scaled to different sizes and used on the same CNN network to see what image size improves the results on that specific network. That approach worked well to experiment and determine the best image scale without having to change the CNN architecture.

The new approach experiments with different CNN architectures and is based on the following idea: images that represent encoding in their pixels rather than natural scenes are prone to information loss when scaling or pooling is applied to them. Scaling and pooling work well on images of natural scenes because a group of pixels in a natural scene could lose one or a few pixels and still, together, look like the exact original scene. However, pooling and scaling may damage the data when the data is images of unnatural scenes, particularly when the images represent encodings.

Pooling reduces the size of input data of the next hidden layer and, as a result, it reduces the number of convolution computations. Removing pooling to improve accuracy, as explained above, increased the size of the input data and the number of convolution computations significantly, which increased training time. In our study, we changed the CNN architecture, using larger hidden layers and filters to handle larger inputs and find the best-performing architecture. We used more computing power to train multiple neuronal networks.

2.2. Data

A positive training set of 46,255 NifH amino acid sequences was obtained by executing the ARBitrator algorithm on a snapshot of the GenBank nr nonredundant protein database taken on

August 31, 2018. A negative training set of 2,013 amino acid sequences was selected from sequences that were returned by submitting ARBitrator's 15 representative sequences as blastp queries against nr and accepting sequences that were not annotated as NifH and were rejected by ARBitrator. Thus, the negative sequences are likely to be accepted by an inaccurate classifier.

After cleaning, preprocessing, and dropping duplicates, the total data was approximately 29k sequences of images. Of the 29K images, about 23,787 represent sequences of DNA that belong to nifH, and approximately 3,000 are a mix of other similar DNA sequences of nifD, nifK and nifN. A common way of splitting the data is by allocating 20% for testing and 80% for training. However, since the negative data set was small, only 2% or 4% was allocated for testing and the rest for training. To avoid having inaccuracies and poor predictions on the negative set, we increased the negative data set. This is, in fact, a big problem that could easily be hidden as having a high rate of correct predictions on the positive set would bias the overall rate to seem better while we perform poorly on the negative data. We utilized other DNA sequences that are not nifH to grow the negative data set as they share old ancestors with nifH before mutating and changing. The new negative dataset was obtained by downloading nifD, nifN, and nifK sequences from FunGene, and then eliminating redundant sequences [46] [47]. After growing the data set to 15869 negative sequences, 80% of the data was allocated to training and 20% to testing.

Table 1: Dataset and splits size

Data Set	**Number of Sequences**
Training Positive Set	19029
Training Negative Set	12695
Testing Positive Set	4758
Testing Negative Set	3174

2.3. Building a Dataset

In this section, we discuss data cleaning and the process of how we represented the data as images. We include examples of the cleaning and transformation that we performed on our data and other examples to clarify the concept of some of the steps.

2.3.1 Data Cleaning

Cleaning the data set included: converting characters to lower case, removing non (a-z) characters and characters that do not represent any of the amino acids in our data set, and eliminating empty sequences and duplicates. Comparisons between sequences in the same file and comparisons against other files, negative vs. positive sets, were done to detect and reduce data errors.

2.3.2 Sequences to Images

Protein sequences are conventionally represented as strings over an alphabet of 20 characters that represent the 20 amino acids. We converted protein sequence strings to ASCII encoding, using the binary representation for each letter. Each letter representation has seven bits that form a row in a 2D image, representing a single sequence. Normalization replaced each bit of value ‘1’ with ‘255’. In order to generate an image from a sequence, each character is converted to seven horizontal black or white squares corresponding to the seven digits of the character’s ASCII representation. White represents 0, and black represents 1. The squares are merged into an nx8 rectangle, where n is the sequence length. Images are scaled to consistent dimensions.

Table 2: example shows the concept of converting characters to ASCII

Characters	A	C	T	G
ASCII Values	065	067	084	071
Binary Values	1000001	1000011	1010100	1000111
Bits	7	7	7	7

Algorithm:

a. Convert each character to the ASCII representation.

b. Convert each ASCII to the corresponding binary representation.

c. Apply Normalization to get values suitable for images 1=> 255

d. Create an image where 255 is a white pixel and 0 is black.

Example:

Consider a sequence of characters: “ACTGACTG…”; this is much smaller than a real sequence; sequences in the data are much longer, and the corresponding images would have many more rows.

a) Convert characters to ASCII

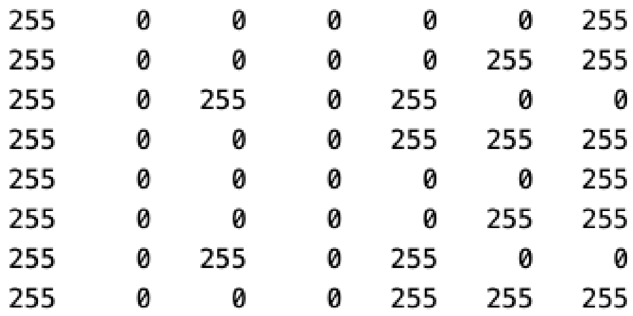
Each sequence is composed of multiple characters and is 1D. When expanding each character in the sequence to the ASCII representation, seven bits in our case, the sequence becomes a 2D matrix

b) Matrix of dimensions 8 x 7

$$\begin{matrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1
\end{matrix}$$

For a real example of a sequence with hundreds of amino acids, the matrix would have seven columns and hundreds of rows. We will see later how neighboring information would be captured by having the rows trained together in a network that captures neighboring information like a CNN.

c) Replace all 1’s with 255’s



```
255     0     0     0     0     0   255
255     0     0     0     0   255   255
255     0   255     0   255     0     0
255     0     0     0   255   255   255
255     0     0     0     0     0   255
255     0     0     0     0   255   255
255     0   255     0   255     0     0
255     0     0     0   255   255   255
```

Figure 1: input example encoded in ASCII and scaled

d) Create image

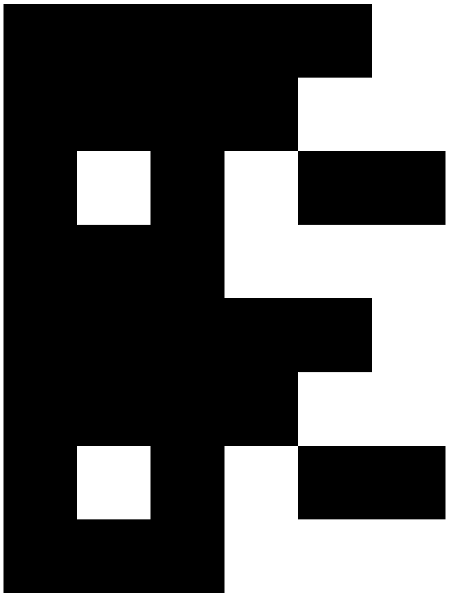


Figure 2: 8 X 7 input Image

Example (2):

Consider a sequence of characters: “GTCA …”;

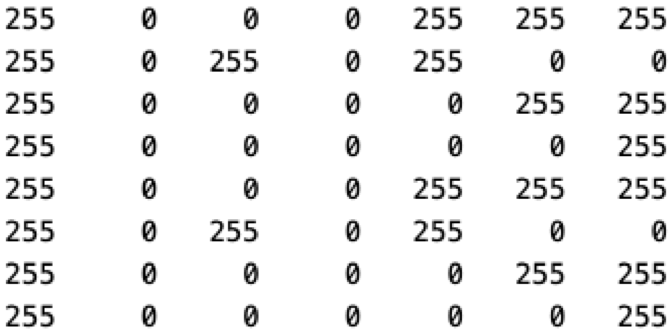
a) Convert characters to ASCII

Each sequence is composed of multiple characters and is 1D. when expanding each character in the sequence to the ASCII representation, seven bits in our case, the sequence becomes a 2D matrix

b) Matrix of dimensions 8 x 7

$$\begin{matrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1
\end{matrix}$$

c) Replace all 1’s with 255’s



```
255     0     0     0   255   255   255
255     0   255     0   255     0     0
255     0     0     0     0   255   255
255     0     0     0     0     0   255
255     0     0     0   255   255   255
255     0   255     0   255     0     0
255     0     0     0     0   255   255
255     0     0     0     0     0   255
```

Figure 3: input example (2) encoded in ASCII and scaled

d) Create image

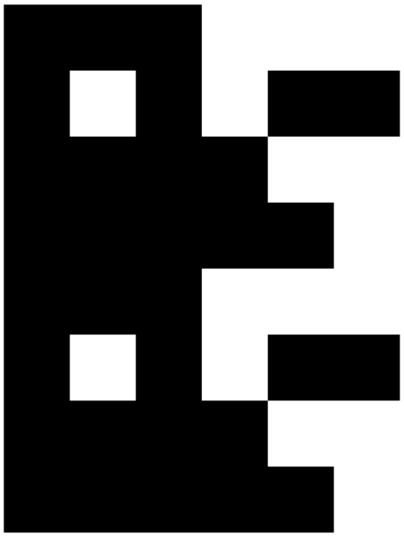


Figure 4: 8 X 7 input Image for example (2)

Like the previous example, this is an image of a matrix consisting of seven columns and many rows. However, an image of a matrix generated from a real sequence in our data has a width of seven pixels and hundreds of rows, making it very long. This will impact the structure of any neural network needed to train these images.

3. CNN Background

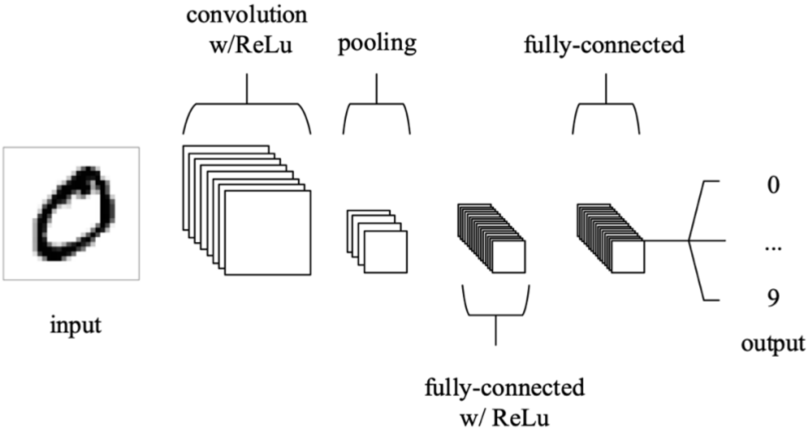
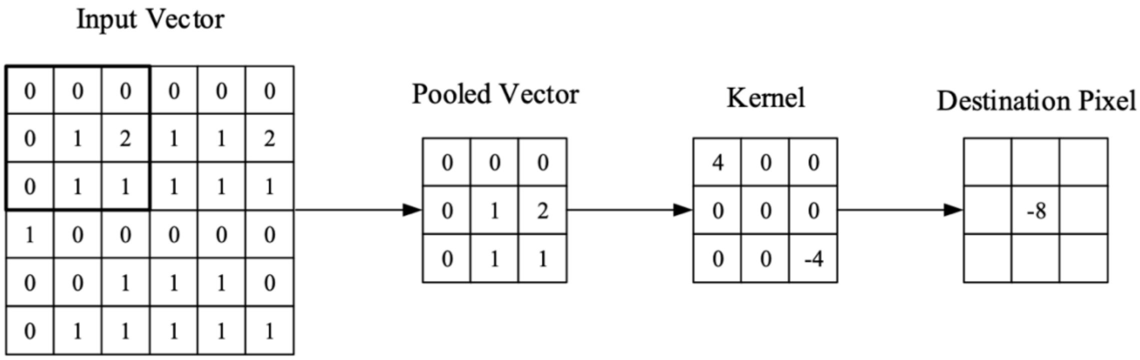


Figure 5: A Simple CNN architecture, comprised of just five layers [39]

CNNs have three types of layers:

- Convolutional layer
- Pooling layer
- Fully-connected layer



Input Vector

0	0	0	0	0	0
0	1	2	1	1	2
0	1	1	1	1	1
1	0	0	0	0	0
0	0	1	1	1	0
0	1	1	1	1	1

Pooled Vector

0	0	0
0	1	2
0	1	1

Kernel

4	0	0
0	0	0
0	0	-4

Destination Pixel

	-8	

Figure 6: The process of convolution [39]

How CNNs work:

1. **The Input layer** has the data representation as a tensor (matrix).
2. **The Convolutional layer** applies the convolution operation on input tensors using weights (neural network specific values). Also, it uses activation functions to add linearity into each layer's output.
3. **The pooling layer** reduces the number of pixels by a process like averaging to have smaller input size.
4. **A dense fully-connected layer** produces probability scores assigned to classes of a classification problem.

Input layer

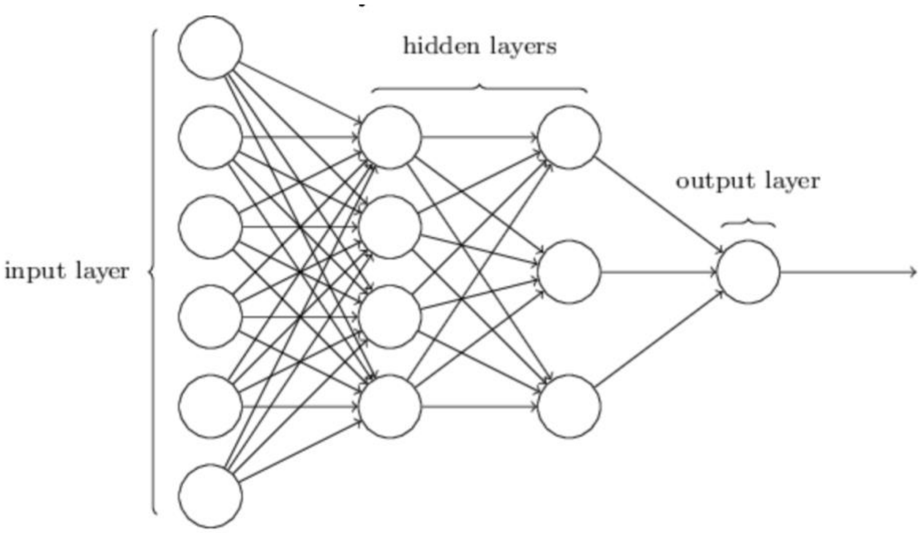


Figure 7: illustration of an input layer matching the input size [42]

The input layer is the first layer that takes the input data. It must match the data size of input after any data pre-processing.

Convolutional Layer

Convolution mainly depends on kernels that are applied to images. These kernels, also called filters, are matrices that are usually much smaller than the size of the input. Convolution calculates the value of an area scanned by a kernel by calculating the value of the central pixel. It multiplies the values of the neighboring pixels by the weights from the kernel and then adds them together. This layer can be tweaked by changing the below variables:

- Depth
- Stride
- Padding

Depth

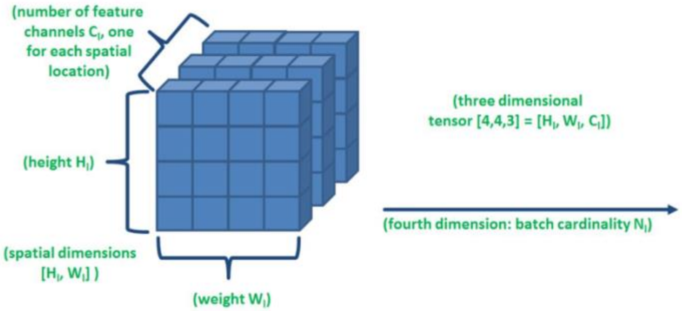

(number of feature channels C_l, one for each spatial location)
(three dimensional tensor [4,4,3] = [H_l, W_l, C_l])
(height H_l)
(fourth dimension: batch cardinality N_l)
(spatial dimensions [H_l, W_l])
(weight W_l)

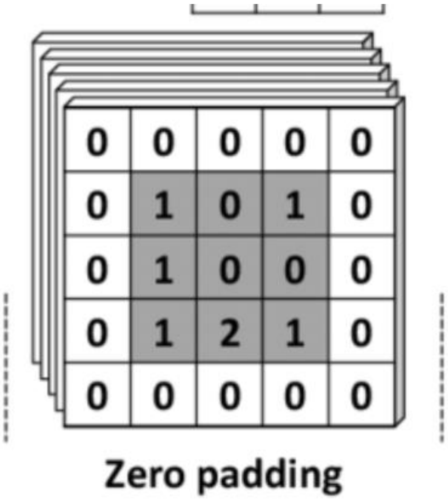
Figure 9: Example showing depth of a tensor

Depth is also known as the number of channels; most commonly, it is the 3rd dimension for 2D images that makes RGB. A 2D filter is applied on a channel to scan all of its pixels, and then it is applied to the next channel.

Stride

Stride is an additional part of the image that is included in a single convolution operation using the filter to cover more parts of the image, making things simpler, easier and faster.

Zero-padding



0	0	0	0	0
0	1	0	1	0
0	1	0	0	0
0	1	2	1	0
0	0	0	0	0

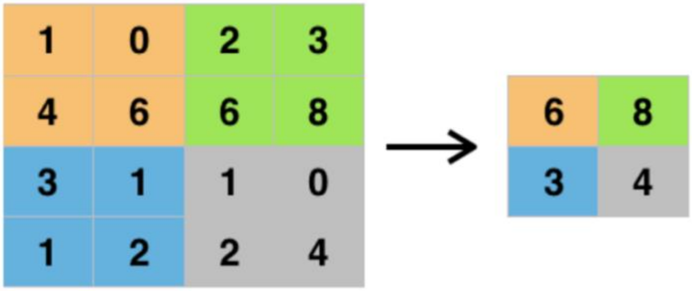
Figure 8: Example of zero padding added to an image [44]

As its name implies, it includes padding the image with zero value pixels to change to allow using filters on the edges.

$$Output\ Shape = \frac{Input\ Shape + 2\ Padding}{Stride + 1}$$

This formula shows how the shape changes, making the output of a layer smaller than its input.

Pooling Layer



1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

Figure 9: Pooling process of 16x16 image to a 4x4 image[1]

Pooling layers are very commonly used to reduce the size of a layer's input and make calculations simpler each time they are used. They are an approximation for a group of pixels that results in a single pixel, which holds the information of all the group pixels. The common ways of pooling are maximum and averaging. Maximum pooling replaces a part of the image, nine pixels, with the maximum value of the existing ones. Average pooling replaces a group of pixels with the average of the pixels' values. It is then applied to the next part of the image, the next nine pixels.

Pooling reduces the number of pixels in an image while somewhat preserving the overall information. For example, a photo of a natural scene of a tree would still look the same and preserve all relative attributes of it.

[1] https://deepai.org/machine-learning-glossary-and-terms/max-pooling

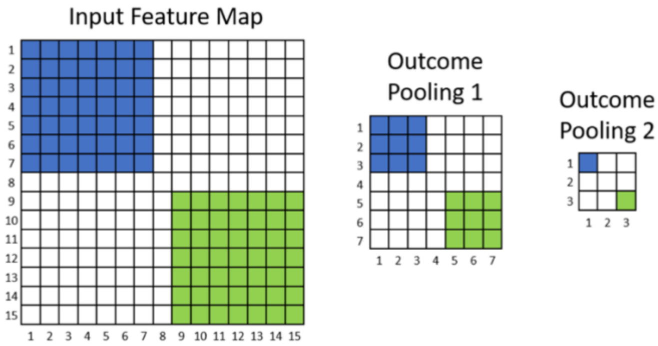


Figure 10: shows how pooling could be very effective in reducing the size of the data [43]

Dense - Fully connected layer

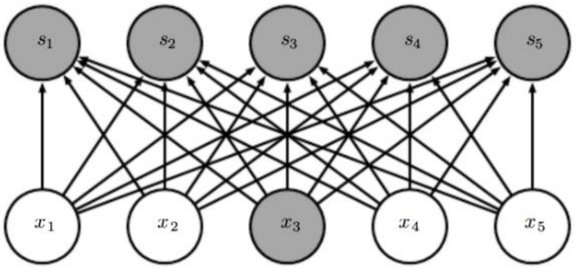


Figure 11: Neurons and the connections between them in a fully connected layer [41]

Fully connected layers allow connections from all input neurons to all activation nodes; They are usually placed towards the end of the network (last few layers, for example).

4. Experiment

The motivation of our experiment is to achieve higher accuracy than what we had before, including improving on false positive and false negative metrics. Different approaches and tuning strategies were tested, including questioning if scaling or pooling could erase some of the information carried in the images representing our genetic data.

Scaling and pooling are widespread and intuitive to use on images as most images that scientists study are images of natural scenes. In natural scenes, as mentioned in previous sections, nearby pixels are similar. Replacing two or a group of neighboring pixels by one is a good idea to reduce the size and relatively preserve the data (or the scene). However, encodings of genetic sequences hide details within the difference between two consecutive pixels. Averaging them could erase the information being studied and learned by the next CNN layer.

Since genetic data is usually long sequences, long filters could play a better role in capturing patterns that depend on a relationship between nucleotides positioned far from each other. These patterns would be detected in earlier layers towards the beginning of the neural network when using long filters

4.1. Experimental Model

Eliminating pooling was one of the main characteristics of all neural networks that did well. However, pooling usually helps reduce the dimensions of the image each time it goes through a layer. Without pooling, the reduction is smaller, and we might end up with a large input size for the last dense layer, which adds overhead to training performance. We used very long convolution filters in some cases; they are around 30% of the length of a sequence. Having long filters in neural networks made for genetic data helps reduce the size of the data and could have another interesting benefit that improves the classification quality.

We tried three different model architectures, two-layer, three-layer, and four-layer models. The first layer has more neurons, and the number of neurons is less for deeper layers because of the convolution. The input layer is a 1000 neuron layer as it has to match the input size. We were able to make all inputs of the same size by padding all of them to a unified size of 1000. Below is a detailed breakdown of the architecture of the three models.

2-Layer CNN – no pooling - long filters

After the input layer, there are two convolution layers; each has a relu activation layer. A dense layer (fully connected layer) is at the end of the neural network with a sigmoid activation function.

- CNN Layer 1 with filters (4,99,4)
- activation 1 relu
- CNN Layer 2 with filters (6,33,1)
- activation 2 relu
- dense layer activation sigmoid

3-Layer CNN – no pooling - long filters

After the input layer, there are three convolution layers; each has a relu activation layer. A dense layer (fully connected layer) is at the end of the neural network with a sigmoid activation function.

- CNN Layer 1 filters (4,99,4)
- activation 1 relu

- CNN Layer 2 filters (6,33,1)
- activation 2 relu
- CNN Layer 3 filters (6,9,1)
- activation 3 relu
- dense layer activation sigmoid

4-Layer CNN – no pooling - long filters

After the input layer, there are four convolution layers; each has a relu activation layer. A dense layer (fully connected layer) is at the end of the neural network with a sigmoid activation function.

- CNN Layer 1 filters (4,99,4)
- activation 1 relu
- CNN Layer 2 filters (6,33,1)
- activation 2 relu
- CNN Layer 3 filters (6,9,1)
- activation 3 relu
- CNN Layer 4 filters (12,9,4)
- activation 4 relu
- dense layer activation sigmoid

4.2. Building CNN Neural Network

The two model architectures that gave the best results and stood out were the two architectures of two-layers and three-layers CNNs. Below are graphs to show a visual representation of these two well-performing networks.

Optimizers are the part responsible for the way we choose to update the weights using a learning rate. We tried a few optimizers like Adam and AdaDelta. Adam gave better results, so we chose it for our models of different sizes.

2-Layer Neural network architecture

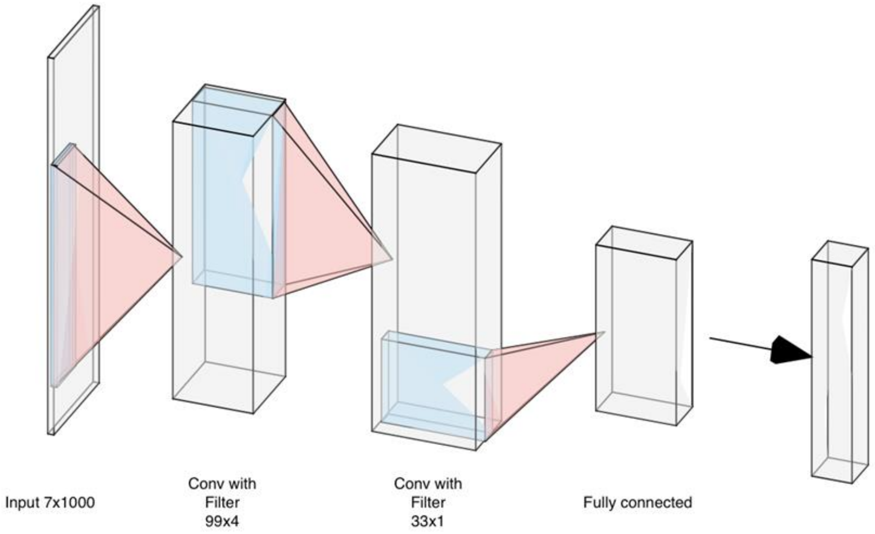


Figure 12: Neural network architecture - 2 Layers

3 Layer Neural Network Architecture

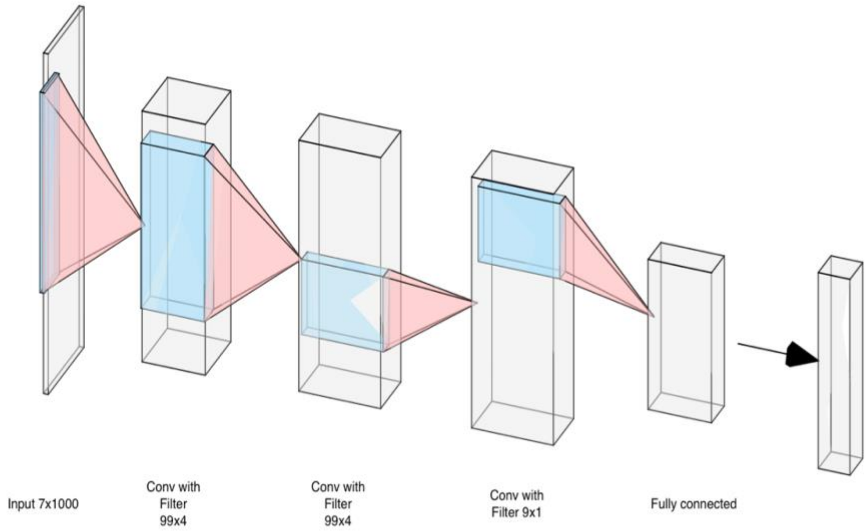


Figure 13: Neural network architecture - 3 Layers

5. Results

5.1. Definitions

Accuracy: The number of successful predictions out of the total number of predictions

FN: The number of wrong predications happening on the negative dataset.

FP: The number wrong predictions happening on the positive dataset.

TP: The number correct predictions happening on the positive dataset

TN: The number correct predictions happening on the Negative dataset

F1 Score = $\frac{TP}{TP\ +0.5(FP+FN)}$

FP rate = $\frac{FP}{FP+TN}$

5.2. Table of Results

Table 3 shows the results of our experiments including loss values, accuracy and error metrics for the different neural networks used, and for different number of epochs.

Table 3: Results and accuracy using CNNs, high accuracy and very low FP and FN

Num Of Layers	Epochs	Loss	Accuracy	F1 Score	FP Rate	FN	FP	TN	TP
2	10	0.018054323212949214	0.9971	0.9969	0.0018	0.0053	0.0013	0.9947	0.9987
2	20	0.021785734529515145	0.9974	0.9973	0.0025	0.0038	0.0017	0.9962	0.9983
2	30	0.0293257650276357	0.9977	0.9976	0.0028	0.0028	0.0019	0.9972	0.9981
2	40	0.02341803015545418	0.9974	0.9973	0.0018	0.0044	0.0013	0.9956	0.9987
3	10	0.024774591952239078	0.9971	0.9969	0.0018	0.0053	0.0013	0.9947	0.9987
3	20	0.02144932239005168	0.9967	0.9965	0.0009	0.0072	0.0006	0.9928	0.9994
3	30	0.03910824464072542	0.9977	0.9976	0.0022	0.0035	0.0015	0.9965	0.9985

3	40	0.04172093846161687	0.9964	0.9963	0.0022	0.0066	0.0015	0.9934	0.9985
4	10	0.01460016401260938	0.9972	0.9971	0.0022	0.0047	0.0015	0.9953	0.9985
4	20	0.02004776849328899	0.9971	0.9969	0.0034	0.0038	0.0023	0.9962	0.9977
4	30	0.019997932241011568	0.9964	0.9963	0.0034	0.0053	0.0023	0.9947	0.9977

6. Discussion

The experiment in this study demonstrated that the highest accuracy is achieved by a 2-layer CNN trained for 30 epochs. It gave excellent results on the genetics data set studied in this research, low FP and FN (FP = 0.0019, FN= 0.0028) (see Table 3).

Our previous experiment tried different input image sizes to find the most optimized one to allow a specific neural network to have the best possible accuracy. The limitation was that not many of the CNN architectures were tried to see which one fits better. Additionally, the input data itself had some information loss by the time it got to the next CNN layer due to pooling. We experimented with different CNN architectures. The CNNs in our study are tweaked to better fit the problem of classifying genetic sequences mainly by respecting the fact that the images represent the encoding of sequences.

The previous studies by Shinde and Heller presented two pooling layers in the CNN, synthesizing all the feedback within a pixel's adjacent region by analyzing the statistical characteristics of the pooled k-pixel region instead of a single pixel [40]. For the layer following a pooling layer, the input parameters are reduced by a factor of approximately k, improving statistical efficiency and reducing storage requirements for the parameters. This approach usually works well for natural scene images. However, as mentioned before, encodings of genetic sequences might be hiding details, particularly between two consecutive pixels. Thus, by removing pooling, our experiment achieved higher accuracy. Further experiments could determine the costs and benefits of pooling.

7. Conclusion

This study used a novel approach to amino acid sequence classification using a Convolutional Neural Network analyzing images derived from sequence strings. The 2-layer architecture of the experiment, coupled with a 30-epoch training regime, produced the best classifier.

The results presented here suggest that this model is a viable alternative to the ARBitrator classifier. While ARBitrator's accuracy is higher, ARBitrator requires significantly more initial setup, and its accuracy is probably acceptable for most applications. This study introduced an approach that may be especially useful for creating classifiers for sequences other than NifH, for example, NifD and NifK, which are the other components of nitrogenase.

References

[1] K. A. Frazer, “Cross-Species Sequence Comparisons: A Review of Methods and Available Resources,” Genome Research, vol. 13, no. 1, pp. 1–12, Jan. 2003, doi: 10.1101/gr.222003.

[2] M. D. Wilkerson, S. D. Schlueter, and V. Brendel, “yrGATE: a web-based gene-structure annotation tool for the identification and dissemination of eukaryotic genes,” Genome Biol, vol. 7, no. 7, p. R58, 2006, doi: 10.1186/gb-2006-7-7-r58.

[3] J. C. Wooley and Y. Ye, “Metagenomics: Facts and Artifacts, and Computational Challenges,” Journal of Computer Science and Technology, vol. 25, no. 1, pp. 71–81, Jan. 2010, doi: 10.1007/s11390-010-9306-4.

[4] V. M. Markowitz et al., “IMG/M 4 version of the integrated metagenome comparative analysis system,” Nucleic Acids Research, vol. 42, no. D1, pp. D568–D573, Jan. 2014, doi: 10.1093/nar/gkt919.

[5] S. Ratnasingham and P. D. N. Hebert, “BARCODING: bold: The Barcode of Life Data System (http://www.barcodinglife.org): BARCODING,” Molecular Ecology Notes, vol. 7, no. 3, pp. 355–364, Jan. 2007, doi: 10.1111/j.1471-8286.2007.01678.x.

[6] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, “GenBank,” Nucleic Acids Research, vol. 37, no. Database, pp. D26–D31, Jan. 2009, doi: 10.1093/nar/gkn723.

[7] H. J. Tripp, I. Hewson, S. Boyarsky, J. M. Stuart, and J. P. Zehr, “Misannotations of rRNA can now generate 90% false positive protein matches in metatranscriptomic studies,” Nucleic Acids Research, vol. 39, no. 20, pp. 8792–8802, Nov. 2011, doi: 10.1093/nar/gkr576.

[8] B. K. Burgess and D. J. Lowe, “Mechanism of Molybdenum Nitrogenase,” Chemical Reviews, vol. 96, no. 7, pp. 2983–3012, Jan. 1996, doi: 10.1021/cr950055x.

[9] D. Bombar, R. W. Paerl, and L. Riemann, “Marine Non-Cyanobacterial Diazotrophs: Moving beyond Molecular Detection,” Trends in Microbiology, vol. 24, no. 11, pp. 916–927, Nov. 2016, doi: 10.1016/j.tim.2016.07.002.

[10] J. C. Gaby and D. H. Buckley, “A global census of nitrogenase diversity: nifH gene diversity,” Environmental Microbiology, vol. 13, no. 7, pp. 1790–1799, Jul. 2011, doi: 10.1111/j.1462-2920.2011.02488.x.

[11] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler, “Hidden Markov Models in Computational Biology: Applications to Protein Modeling UCSC-CRL-93-32,” p. 62, Aug. 1993.

[12] J. C. Gaby and D. H. Buckley, “A comprehensive aligned nifH gene database: a multipurpose tool for studies of nitrogen-fixing bacteria,” Database, vol. 2014, Jan. 2014, doi: 10.1093/database/bau001.

[13] P. Heller, H. J. Tripp, K. Turk-Kubo, and J. P. Zehr, “ARBitrator: a software pipeline for on-demand retrieval of auto-curated nifH sequences from GenBank,” Bioinformatics, vol. 30, no. 20, pp. 2883–2890, Oct. 2014, doi: 10.1093/bioinformatics/btu417.

[14] S. Altschul, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” Nucleic Acids Research, vol. 25, no. 17, pp. 3389–3402, Sep. 1997, doi: 10.1093/nar/25.17.3389.

[15] A. Marchler-Bauer et al., “CDD: NCBI’s conserved domain database,” Nucleic Acids Research, vol. 43, no. D1, pp. D222–D226, Jan. 2015, doi: 10.1093/nar/gku1221.

[16] P. Heller, J. Casaletto, G. Ruiz, and J. Geller, “A database of metazoan cytochrome c oxidase subunit I gene sequences derived from GenBank with CO-ARBitrator,” Scientific Data, vol. 5, p. 180156, Aug. 2018, doi: 10.1038/sdata.2018.156.

[17] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 86. 2278 - 2324. 10.1109/5.726791.

[18] J.P. Zehr, Capone, D.G. Problems and promises of assaying the genetic potential for nitrogen fixation in the marine environment. Microb Ecol 32, 263–281 (1996). https://doi.org/10.1007/BF00183062

[19] Z. Zhang, "Improved Adam Optimizer for Deep Neural Networks," 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 2018, pp. 1-2, doi: 10.1109/IWQoS.2018.8624183.

[20] V. I. Jurtz, Johansen, A. R., Nielsen, M., Almagro Armenteros, J. J., Nielsen, H., Kaae Sønderby, C., Winther, O., & Kaae Sønderby, S. (2017). An introduction to Deep learning on biological sequence data - Examples and solutions. Bioinformatics, 33(22), 3685-3690. https://doi.org/10.1093/bioinformatics/btx531

[21] W. S. Ahmed and A. a. A. Karim, "The Impact of Filter Size and Number of Filters on Classification Accuracy in CNN," 2020 International Conference on Computer Science and Software Engineering (CSASE), Duhok, Iraq, 2020, pp. 88-93, doi: 10.1109/CSASE48920.2020.9142089.

[22] A. Krizhevsky, I. Sutskever, GE Hinton: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems: 2012. 1097-1105. 21.

[23] Y. Kim: Convolutional neural networks for sentence classification. arXiv preprint arXiv:14085882 2014.

[24] J. Eickholt, J. Cheng: Predicting protein residue–residue contacts using deep networks and boosting. Bioinformatics 2012, 28(23):3066-3072

[25] J. Schmidhuber, ‘‘Deep learning in neural networks: An overview,’’ Neural Netw., vol. 61, pp. 85–117, Jan. 2015

[26] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, ‘‘Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning,’’ Nature Biotechnol., vol. 33, no. 8, pp. 831–838, 2015.

[27] Y. S. Vang and X. Xie, ‘‘HLA class I binding prediction via convolutional neural networks,’’ Bioinformatics, vol. 33, no. 17, pp. 2658–2665, 2017

[28] J. Zhou and O. G. Troyanskaya, ‘‘Predicting effects of noncoding variants with deep learning-based sequence model,’’ Nature Methods, vol. 12, no. 10, pp. 931–934, 2015.

[29] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford, ‘‘Convolutional neural network architectures for predicting DNA–protein binding,’’ Bioinformatics, vol. 32, no. 12, pp. i121–i127, 2016

[30] D. R. Kelley, J. Snoek, and J. L. Rinn, ‘‘Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks,’’ Genome Res., vol. 26, no. 7, pp. 990–999, 2016.

[31] P. Arena, L. Fortuna and L. Occhipinti, "A CNN algorithm for real time analysis of DNA microarrays," in IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 49, no. 3, pp. 335-340, March 2002, doi: 10.1109/81.989167.

[32] Z. Abbas, H. Tayara, and K. T. Chong 2021. "4mCPred-CNN—Prediction of DNA N4-Methylcytosine in the Mouse Genome Using a Convolutional Neural Network" Genes 12, no. 2: 296. https://doi.org/10.3390/genes12020296

[33] C. Renzhi, C. Freitas, L. Chan, M. Sun, H. Jiang, and Z. Chen. 2017. "ProLanGO: Protein Function Prediction Using Neural Machine Translation Based on a Recurrent Neural Network" Molecules 22, no. 10: 1732. https://doi.org/10.3390/molecules22101732

[34] M. Kulmanov, R. Hoehndorf, DeepGOPlus: improved protein function prediction from sequence, Bioinformatics, Volume 36, Issue 2, 15 January 2020, Pages 422–429, https://doi.org/10.1093/bioinformatics/btz595

[35] L. N. Nguyen V. 2019. SNARE-CNN: a 2D convolutional neural network architecture to identify SNARE proteins from high-throughput sequencing data. PeerJ Computer Science 5:e177 https://doi.org/10.7717/peerj-cs.177

[36] N. Srikanth et al. “Automated Quantification of DNA Damage via Deep Transfer Learning Based Analysis of Comet Assay Images.” Vol. 11139. SPIE, 2019. 111390Y–111390Y–7. Web.

[37] M. Tropea and G. Fedele, "Classifiers Comparison for Convolutional Neural Networks (CNNs) in Image Classification," 2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2019, pp. 1-4, doi: 10.1109/DS-RT47707.2019.8958662.

[38] A. Sharma, E. Vans, D. Shigemizu et al. DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture. Sci Rep 9, 11399 (2019). https://doi.org/10.1038/s41598-019-47765-6

[39] K. O'Shea., R. Nash. "An Introduction to Convolutional Neural Networks" https://arxiv.org/pdf/1511.08458.pdf

[40] I. Shinde. "Nitrogenase Iron Protein Detection using Neural Network ", May 2019

[41] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MT Press, 2016

[42] M. Nielsen, "using neural nets to recognize had written digits" Dec, 26th 2019

http://neuralnetworksanddeeplearning.com/chap1.html

[43] M. Sousa, "Visualizing the Fundamentals of Convolutional Neural Networks" https://towardsdatascience.com/visualizing-the-fundamentals-of-convolutional-neural-networks-6021e5b07f69

[44] S. Kamrava, P. T. Sahimi, "Research Gate" https://www.researchgate.net/figure/llustration-of-convolutional-filter-computation-for-an-input-data-after-zero-padding_fig4_336908534

[45] "Computer Hope", March 13th 2021 https://www.computerhope.com/issues/ch001632.htm

[46] A. Rez "nifH classification", 2019, https://github.com/amerrez/nifh-classification

[47] Fish, J. A., B. Chai, Q. Wang, Y. Sun, C. T. Brown, J. M. Tiedje, and J. R. Cole, 2013. FunGene: the Functional Gene Pipeline and Repository. Front. Microbiol. 4: 291.