CS230

Segmentation of Modern Human genotypes based on ancient DNA

Pierre J. M. Chambon
ICME
Stanford University
pchambon@stanford.edu

Rebecca L Hwang
Department of Electrical Engineering
Stanford University
rlhwang@stanford.edu

Abstract

DNA ancestry analysis helps identify the history of populations as well as genes and variants that are associated with differing levels of resistance to various diseases and pathologies. We propose NeanderNet, a deep learning model capable of identifying subsequences inherited from Homo Neanderthalensis, and which can be used in other ancient DNA classification/segmentation tasks. We combine past related approaches into a new architecture, a CNN-BiLSTM, and compute the first results for this model on real genotype data from the UCSC Genomics Institute and the 1000 Genomes Project.

1 Introduction

The study of adaptive introgression provides a rich database of evolutionary history and is of great significance to understanding our modern health issues [5]. For example, recent studies have shown the role ancient DNA plays with COVID-19 [7] as well as hepatitis [10]. However, this field of study could be greatly advanced with the continued integration of deep learning.

The input to our algorithm are DNA sequences; we then use a CNN, LSTM or their combination as a CNN-BiLSTM to determine if the sequence in question is introgressed (Neanderthal, or any other type of ancestor such as Denisovan) or of a modern human.

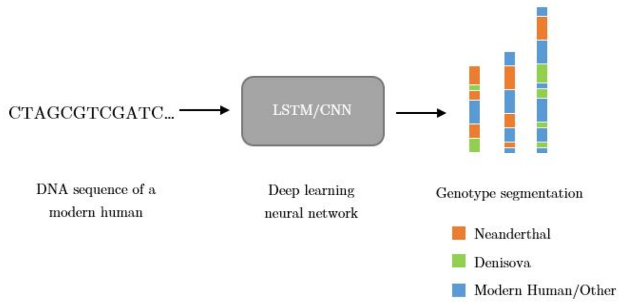


Figure 1: Simple model overview.

2 Related work

There were two major avenues that we wanted to explore – the first was the traditional CNN approach, mentioned in [3]. The second avenue was an LSTM approach; we drew upon various works [4], [2]. In terms of encoding our input data, our basic approaches were similar to [6] in how we either used k-mer embeddings for representing our input data or one-hot encoding. There were some methodologies that we found particularly interesting – i.e. Yin's work [12] was quite unique in how the data was embedded. However, in the hopes of iterating quickly, we decided that this embedding would be an interesting future direction.

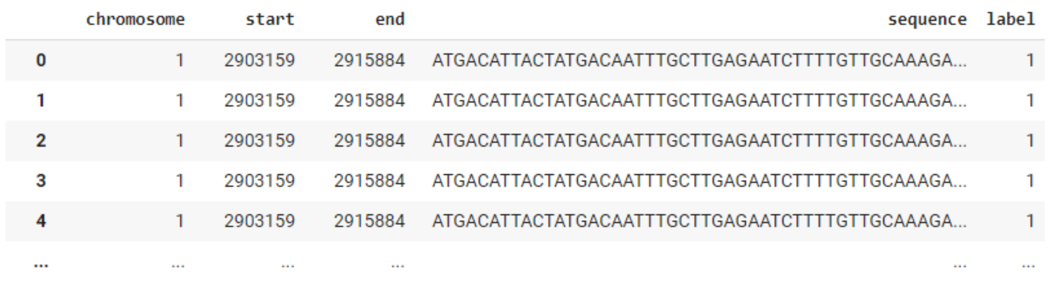
3 Dataset and Features

Thanks to the studies led by 1000 Genomes Project and the University of California, Santa Cruz, multiple samples of ancient DNA are now publicly available on the internet. We identity two main datasets of ancient DNA, coming from:

- subspecies of the Homo genus, such as Homo Sapiens, Homo neanderthalensis and Homo denisova.
- various ancestors from Bronze Age and Middle Age, across different regions and continents.

We therefore choose to develop an approach which could be used across these various datasets, with a model capable of understanding the structure and patterns of the DNA; this final model can then be used on different tasks through transfer learning.

For each of these datasets, we have access to either (1) sequences from the modern human genotype which were identified as belonging to specific ancestors, or (2) sequences directly extracted from old DNA, found on archaeological sites. In the first case, the data is often represented as a list of index pairs, with the starting index and the ending index of the subsequence in the modern human genotype. The datasets we manipulate refer to HG19, one of the reference human genotypes. Some other datasets directly contain sequences of DNA (this is the case when the DNA is extracted from archaeological samples) or alignment files (the dataset only underlines the differences with respect to a reference human genotype).



	chromosome	start	end	sequence	label
0	1	2903159	2915884	ATGACATTACTATGACAATTTGCTTGAGAATCTTTTGTTGCAAAGA...	1
1	1	2903159	2915884	ATGACATTACTATGACAATTTGCTTGAGAATCTTTTGTTGCAAAGA...	1
2	1	2903159	2915884	ATGACATTACTATGACAATTTGCTTGAGAATCTTTTGTTGCAAAGA...	1
3	1	2903159	2915884	ATGACATTACTATGACAATTTGCTTGAGAATCTTTTGTTGCAAAGA...	1
4	1	2903159	2915884	ATGACATTACTATGACAATTTGCTTGAGAATCTTTTGTTGCAAAGA...	1
...	...	...	...	...	...

Figure 2: Snapshot of dataset. The labels depend on the task; in this case the task is determining if the origin of the sequence is from Homo neanderthalensis or not.

In the case of the classification task between different subspecies of the Homo genus, we must designate our 0 label samples to be those from modern humans (Homo Sapiens) without any relation or inheritance from the other subspecies of interest. To do so, we extract random sub-sequences from each chromosome of the reference human genotype, and check that they do not overlap with any sequences inherited from other subspecies. The final sequences also need to be cleaned, as the DNA coding process may fail to identity certain nucleobases (in this case signaled with a 'N' in the DNA sequence).

After going through these steps, we come up with a dataset of approximately 61,860 sequences, each containing 200 nucleotides, with a label 1 if it is inherited from Homo neanderthalensis and 0 if it is not. We perform a 80/20 split into training and testing datasets. Having longer sequences or a bigger dataset would surely give better results, and this could be achieved by directly contacting laboratories such as the USCS Genomics Institute. Nevertheless, we decided to stick with this dataset which allows us to iterate more quickly, with the hopes that finding a good model that performs well could then scale to larger datasets to get even better results.

Before training, we also split the dataset by paying attention to data leakage. Some sequences were almost duplicates (extracted from similar positions on different individuals, and differing only by a few nucleotides). Therefore, we decided each time to put all of them either in the train split or the test split. This would allow us to get a model that is better at generalizing on unseen portions of the DNA.

4 Methods

4.1 1D CNN and 2D CNN

As mentioned in Section 2, most previous works analysing DNA with Deep Learning implemented a CNN architecture. Therefore our initial model was a 1D CNN with 3 layers. We also decided to implement Keras' hyperparameter tuners package, which would save us the trouble of manually searching through various combinations of hyperparameters. Specifically, we implemented the random and hyperband approaches to search for the best dropout and learning rate.

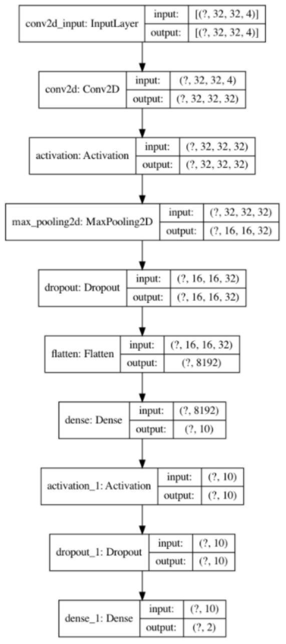


Figure 3: CNN Model with Dropout

We then progressed to using a 2D CNN model - our 2D CNN model is built upon Nikolay Oskolkov's work [1] as the base – we modify the neural network architecture throughout our experiments. In order to represent our sequence as a 2D image, the code takes the n^{th} example in the input ,e.g. ATCAA...CGG (length of 32 nucleotides), and then duplicates this example 32 times forming a 32 by 32 image. This transformation is applied to all of the input examples which we then pass into a CNN with kernel size of (5,5). We note that our model seems to over fit (there is a huge difference in loss between test and train), so we introduced dropout into the model (Figure 3). However, this did not seem to improve the model performance by a respectable amount, see Figure 5, Figure 6. We also used the Adam optimizer, as it is computationally efficient and works well on large datasets. For both CNN models, the input data was encoded using the one hot encoding – because there are 4 nucleotides, the resulting encoding was a 32 by 4 dimensional matrix. For both CNNs we used the binary cross entropy loss function, as indicated below:

$$\ell = -\frac{1}{N}\sum y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i))$$

4.2 BiLSTM

As the DNA data is represented in 1D-sequences, the LSTM/BiLSTM architecture seems to be well-adapted for the problem. DNA sequences can very quickly elongate, thus creating long-range dependencies that the BiLSTM would hardly detect. For this reason before inputting data directly in this model, we add an additional pre-processing step to reduce the length of the input.

We decide to model each sequence of DNA as a text, from which one we can extract subsequences of a fixed length. These subsequences themselves also contain words of a fixed length (i.e. k-mers). This introduces another parameter related to the input data, thereby giving us an additional dimension of control. In addition, we can split DNA sequences into words in several manners. A first possibility is to form a word with any new subsequence of length word_length. Another possibility is to create a new word given an offset related to the previous word. This offset could be fixed to 1, so that each consecutive word only differs in 1 character compared to its direct neighbors. These words can be interpreted as a way to give context to each character of the sequence. We also try the possibility of words of length 1, thus reducing our inputs to sequences of only one character. Finally, when splitting into training and testing data, we make sure to not include sentences from the same DNA text (DNA sample) in both datasets.

The data is then passed through a tokenizer that converts each word to its index in the vocabulary.

The BiLSTM architecture is depicted in Figure 4 [4] [9]. We use a learned embedding that maps each word to a 30-dimensional feature space, before being inputted to the BiLSTM with 30 hidden units. As the model is powerful and has a lot of parameters, we use a strong dropout at the output to reduce the possibility of overfitting. We add two additional linear layers to map the outputs to the number of labels and their probabilities. The model is trained using mini-batch gradient descent and batch size of 512, to reduce the noise of the loss function. We use the binary cross entropy loss, as this is a binary classification task.

The BiLSTM implementation is also optimized to run as fast as possible, though the process still takes time due to the limited possibilities of parallelization. Nevertheless, we try to use GPUs and high-efficiency CPUs of AWS, in order to speed up the learning.

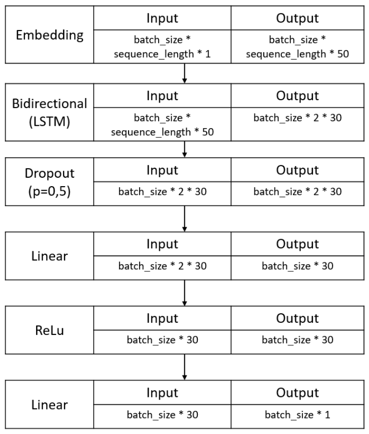


Figure 4: BiLSTM architecture.

4.3 CNN-BiLSTM

The previous models suffer from limitations due to their inherent architecture: the CNN does not completely benefit from the 1D-sequence aspect of the data, and the LSTM suffers from long-range dependencies thus needing an arbitrary preprocessing step. To solve both issues, we try to create a CNN-BiLSTM model, with the goal of taking the best from both worlds and achieving better results.

The CNN-BiLSTM architecture is depicted in Figure 7 [9]. It takes as input the sequence of nucleotides of a DNA portion, passes it through three CNN layers with 1D convolution, 1D max pooling, dropout and batch normalization, then enters a two-layer BiLSTM, and at the end of the model, there is a Dense layer with a sigmoid activation function. We typically start by using the best architectures and hyperparameters found on the isolated CNN and BiLSTM models, before iterating and fine-tuning the new model. The loss function is binary cross-entropy.

In order to quickly iterate, we first start by using the Keras library to build and test various model architectures, and then once we reach a model with good and stable results, we implement it in PyTorch to fine-tune certain parameters. CUDA and GPUs are also useful to parallelize and therefore help us iterate more quickly.

To improve the performance of the model, we try several initializations, in particular the Xavier and He uniform initialization. For the optimizer, we compare the results of Adam and Adam with weight decay, and use momentum to stabilize the training. The learning rate is found with a learning rate finder, and then scheduled following the cyclical/triangular learning rate policy, with various iteration size and decay value over time.

The most difficult part is to control the overfitting of the model, which has a high learning capability. The CNN layers are mainly controlled with kernel regularization and dropout, and the BiLSTM through kernel regularization. We avoid using dropout directly after the BiLSTM as it may affect its performance [13], [11].

5 Experiments/Results/Discussion

The models are evaluated on the binary classification task of determining whether a DNA sequence belongs to Homo neanderthalensis or a modern human. We use the accuracy and F1 score for each label, and as the dataset is balanced we simply average them on the two labels to get a single score. We also provide additional insights by computing the Matthews correlation coefficient on the best models, because it captures the performance of a model on both classes well. To get comparable results, we seed the random functions of the libraries used and compute the different results on the same machine.

Regarding the performance of the CNN, we see from the Figure 5 and Figure 6 that the model is highly overfitting, even with dropout applied. The average accuracy and F1 score on both classes are 0.63 and 0.62, and the Matthews correlation coefficient is 0.25. We implemented the Keras hyperparameter tuning package on the 1D CNN, however we found that the random search approach did not significantly improve performance to justify using it.

Concerning the BiLSTM, we compare the results obtained with different architectures (hidden units, embedding size) and different parameters for the data preprocessing step. The best performances are found to be around 30/50 hidden units, and an embedding size of 30/50 dimensions too. Below and above, the model accuracy decreases.

The preprocessing of the dataset is tested with different word splits. First, we split the data into individual characters, thus inputting into the LSTM monomers. In this case, sentences are extra-long and the LSTM performs poorly due to lack of precision on long-range dependencies. Second, we split the dataset into words which do not overlap. Again, the LSTM performs poorly because each new character has not enough context. The best results are achieved when splitting into several-characters-long words (around 10-12), with an offset between each word smaller than the length of the words. For a word length of 10-12, we use an offset of 1 or 2. For the best LSTM model, the average accuracy and F1 score on both classes are 0.74 and 0.739, and Matthews correlation coefficient is 0.48.

	Character label	Sliding words	Separate words
Class 0	0.49	0.69	0.55
Class 1	0.63	0.74	0.68

	10 hidden units	30 hidden units	100 hidden units
Class 0	0.69	0.77	0.79
Class 1	0.74	0.71	0.68

Table 1: Accuracy for each class for different preprocessings and LSTM configurations.

Finally, the best results are achieved with the CNN-BiLSTM architecture. We try various kernel sizes for the CNN layers, hidden sizes for the BiLSTM layers, and initialization and optimizer configurations. The influence of each choice and the validation loss obtained are summarized in Table 2. The best performance is achieved with three CNN layers with kernel size 20 and 32 channels, max pooling with kernel size 2 and stride 2, dropout with value 0.15, BiLSTM with hidden size 70, learning 0.005 with triangular and exponential decay scheduling. The average accuracy on both classes is 0.75, the average F1 score 0.75, and the corresponding Matthews correlation coefficient is 0.502.

6 Conclusion/Future Work

We introduced NeanderNet, a CNN-BiLSTM architecture capable of understanding the patterns and the content of DNA sequences to perform a classification task related to ancient DNA identification. This architecture combines the proficiencies of both CNNs and LSTMs, which on their own perform worse on the same task and dataset. Our model is capable of classifying subsequences of a modern human DNA on whether they are inherited from Homo Neanderthalensis, and we hope it can perform well on other ancient DNA related tasks, such as local and global ancestry inference.

However, significant work remains in order to obtain viable results for real-world applications. In our future work, we will (1) construct a bigger dataset of higher quality (longer DNA subsequences), so that we can scale and further fine-tune our model; (2) test and adapt the model on other ancient DNA problems to build a good architecture to generally understand the DNA sequences; (3) develop transfer learning approaches, using ULMFit methods for instance [8], to have the model ready to be distributed and used on various smaller tasks and datasets.

7 Contributions

We both contributed equally to literature review and written reports. Pierre cleaned and preprocessed the data, implemented the BiLSTM and at the end the CNN-BiLSTM. Rebecca implemented CNN 1D, CNN 2D, and hyperparameter search.

8 Appendix

CNN-BiLSTM general architecture	BCELoss score
1 CNN/1 LSTM/No LRS/No WD	0.68
3 CNN/2 LSTM/LRS/No WD	0.6
3 CNN/2 LSTM/LRS/WD	0.58

CNN layers architecture	BCELoss score
(10,1)(20,1)(30,2)	0.63
2*(5,1)(5,1)(5,2)	0.66
(20,1)(20,2)(20,2)	0.6
(20,2)(20,2)(20,2)	0.56

Table 2: Binary cross-entropy validation loss for each configuration of the CNN BiLSTM. LRS stands for learning rate scheduler (triangular), WD for weight decay. (x,y) stands for a CNN layer with a convolution of kernel size x and a max pooling of size and stride y.

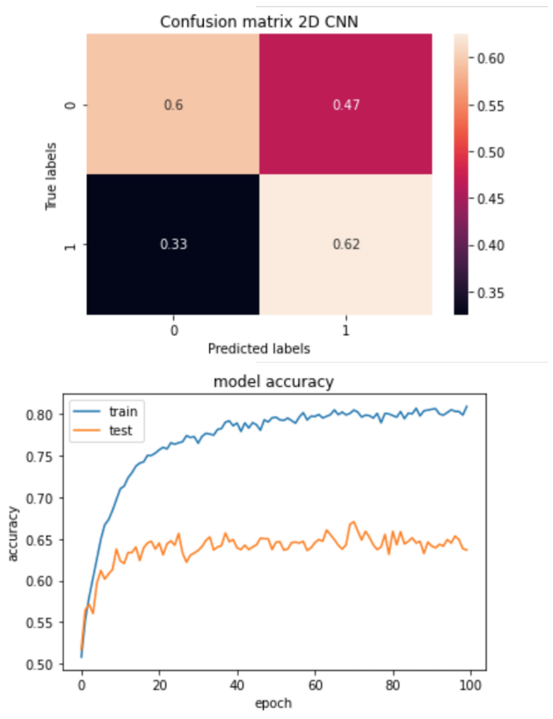


Figure 5: CNN Model Results

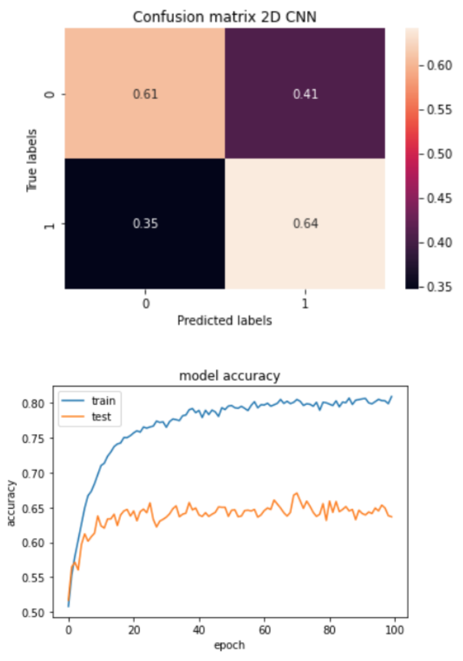


Figure 6: CNN Model with Dropout Results

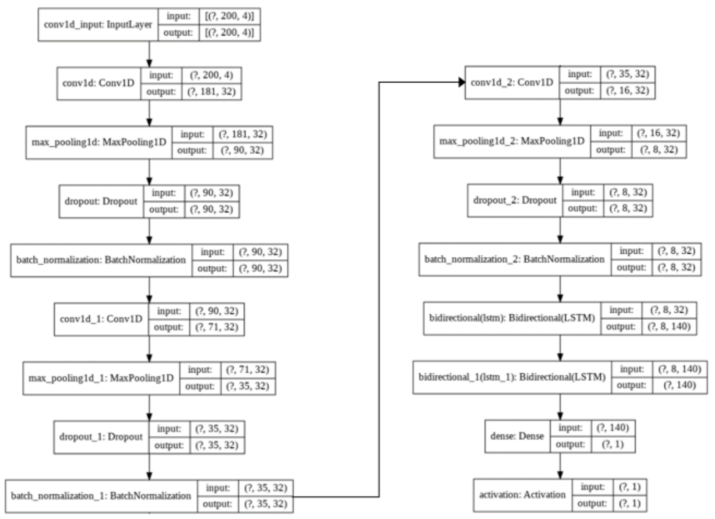

conv1d_input: InputLayer | input: [(?, 200, 4)] | output: [(?, 200, 4)]
conv1d: Conv1D | input: (?, 200, 4) | output: (?, 181, 32)
max_pooling1d: MaxPooling1D | input: (?, 181, 32) | output: (?, 90, 32)
dropout: Dropout | input: (?, 90, 32) | output: (?, 90, 32)
batch_normalization: BatchNormalization | input: (?, 90, 32) | output: (?, 90, 32)
conv1d_1: Conv1D | input: (?, 90, 32) | output: (?, 71, 32)
max_pooling1d_1: MaxPooling1D | input: (?, 71, 32) | output: (?, 35, 32)
dropout_1: Dropout | input: (?, 35, 32) | output: (?, 35, 32)
batch_normalization_1: BatchNormalization | input: (?, 35, 32) | output: (?, 35, 32)
conv1d_2: Conv1D | input: (?, 35, 32) | output: (?, 16, 32)
max_pooling1d_2: MaxPooling1D | input: (?, 16, 32) | output: (?, 8, 32)
dropout_2: Dropout | input: (?, 8, 32) | output: (?, 8, 32)
batch_normalization_2: BatchNormalization | input: (?, 8, 32) | output: (?, 8, 32)
bidirectional(lstm): Bidirectional(LSTM) | input: (?, 8, 32) | output: (?, 8, 140)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (?, 8, 140) | output: (?, 140)
dense: Dense | input: (?, 140) | output: (?, 1)
activation: Activation | input: (?, 1) | output: (?, 1)

Figure 7: CNN-BiLSTM architecture.

References

[1] Deep Learning on Ancient DNA. Reconstructing the Human Past with Deep… | by Nikolay Oskolkov | Towards Data Science.

[2] LSTM to Detect Neanderthal DNA. Long-Memory Neural Networks for Ancient… | by Nikolay Oskolkov | Towards Data Science.

[3] G. Aoki and Y. Sakakibara. Convolutional neural networks for classification of alignments of non-coding RNA sequences. *Bioinformatics*, 34(13):i237–i244, 2018.

[4] J. P. Chiu and E. Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4(2003):357–370, 2016.

[5] E. Y. Durand, J. L. Mountain, J. M. Macpherson, E. Y. Durand, C. B. Do, J. L. Mountain, and J. M. Macpherson. White Paper 23-16 Authors : Ancestry Composition : A Novel , Ef ficien t Pipeline for Ancestry Deconvolution. 2014.

[6] G. Eraslan, Avsec, J. Gagneur, and F. J. Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389–403, 2019.

[7] A. Ganna, T. G. Unit, and M. General. The COVID-19 Host Genetics Initiative, a global initiative to elucidate the role of host genetic factors in susceptibility and severity of the SARS-CoV-2 virus pandemic. *European Journal of Human Genetics*, 28(6):715–718, 2020.

[8] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, 1:328–339, 2018.

[9] S. Hu, R. Ma, and H. Wang. An improved deep learning method for predicting DNA-binding proteins based on contextual features in amino acid sequences. *PLoS ONE*, 14(11):1–21, 2019.

[10] B. Krause-Kyora, J. Susat, F. M. Key, D. Kühnert, E. Bosse, A. Immel, C. Rinne, S. C. Kornell, D. Yepes, S. Franzenburg, H. O. Heyne, T. Meier, S. Lösch, H. Meller, S. Friederich, N. Nicklisch, K. W. Alt, S. Schreiber, A. Tholey, A. Herbig, A. Nebel, and J. Krause. Neolithic and medieval virus genomes reveal complex evolution of hepatitis B. *eLife*, 7:1–15, 2018.

[11] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal. Zoneout: Regularizing rNNs by randomly preserving hidden activations. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–11, 2017.

[12] B. Yin, M. Balvert, D. Zambrano, M. Sander, and C. Wiskunde. a N Image Representation Based Convolutional. pages 1–14, 2018.

[13] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. (2013):1–8, 2014.