CSCI 4622: Machine Learning Project

MNIST Digit Classification Using LSTM RNNs: Representing Images as Time Series Using Hilbert Curves

Beckett Hyde, Blake Hamilton

Spring 2023

1 Problem Space

Constructing an algorithm that can interpret images is a manually intractable problem. The content is messy, inconsistent, and context-dependent. For this reason, handwritten digit labeling tasks, and image labeling tasks more generally, are common applications of Machine Learning technology. Neural Networks, particularly Convolutional Neural Networks (CNNs), have been applied in the past to image labeling problems with much success [5] [1]. These networks apply various filters, called "convolutions", to the image and use the features identified in these filters to make a decision.

Recurrent Neural Networks (RNNs) are another type of artificial neural network that are useful for analysing time series data because of their ability to keep state over time [7]. Time-series input is fed into the network slowly as opposed to all at once; the neural network takes the current input, as well as its memory of past input, and computes a solution. This solution is often a prediction of future behavior or a categorization of the current behavior.

Motivated by certain emerging ML technologies that only work on time series, like Reservoir Computers and others, we explored various ways of representing images as time-series and applying RNNs, specifically Long Short-Term Memory (LSTM) networks, to digit classification tasks instead of the usual CNNs. We used Hilbert space-filling curves to map the 2-dimensional input of each digit to a 1-dimensional time-series and trained a classifier RNN on the input. We compared the performance of this network to a traditional CNN as a baseline.

There are many documented successful solutions to recognizing handwritten digits using CNNs: Keras (TensorFlow) even provides a walk-through on how to interpret their MNIST dataset (the dataset we are using) with a CNN approach. We are attempting to completely restructure the neural network approach for this problem, and the approach we are taking has not been documented anywhere else. We chose to put this twist on common image recognition approaches using a simple, common dataset because it is straightforward to compare our results to the already successful approaches.

2 Dataset

We used the Keras MNIST Digits Classification Dataset [8]. This dataset includes $70,000$ digit images, $10,000$ of which are test entries. Each entry has $28 \cdot 28 = 784$ features, as each digit image is a square with 28 pixel edge lengths. We consider this, and the set of 14×14 images derived from the larger set by averaging blocks of 2×2 pixels. Each feature has a number range between 0 to 255 (normalized to $[0, 1]$) with the values corresponding to the pixel darkness (as the images are greyscale). The labels are the standard base ten digits, $d = \{1, 2, ..., 9\}$. This dataset is a popular handwritten digit set, and it has been used in many applications.

3 Approach

3.1 Hilbert Space-Filling Curves

To represent an image as a time series, each pixel needs to be lined up sequentially in one dimension. Dimension reduction from two-dimensional space to one-dimensional space while retaining relative location information is difficult. The fact that "pixel a is one unit down and two units left from pixel b" is intrinsically two-dimensional. Therefore, mapping some $N \times N$ discrete space of arbitrary N to a one-dimensional space M of size $|M| = N^2$ is guaranteed to eliminate the distance relationships between some of the points because the "manifold" is fundamentally different.

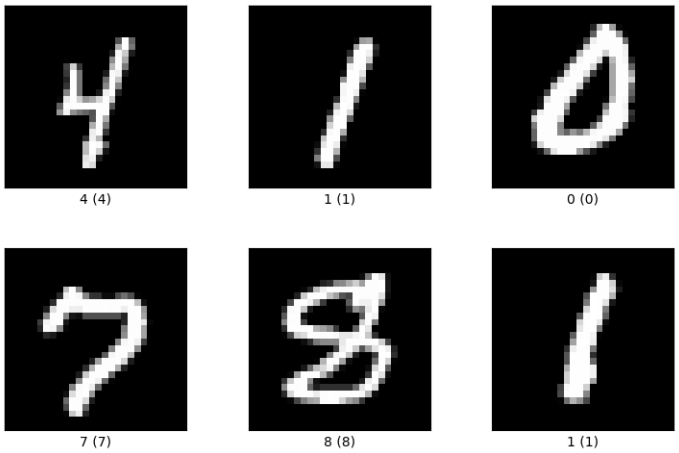


Figure 1: Examples of the MNIST Digits dataset [8]

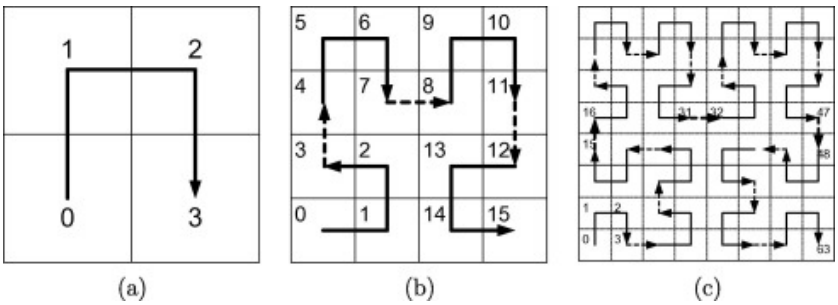


Figure 2: Visual Representation of a Hilbert Space-filling Curve [2]

Hilbert space-filling curves are a way to map every point in $N \times N$ to a point in M that may preserve more of these relationships than other maps [2] [3] (Figure 2) . In the naive approach to dimension reduction, we would take every point in $N \times N$ sequentially, row by row, and insert them into M. This would preserve distance relationships between points in the same row, but would eliminate or severely distort that relationship for points close together in the same or nearby columns. Hilbert curves distort both dimensions of relative location less by "winding" through the space, increasing the information value of data points' relative location, an important component of time series.

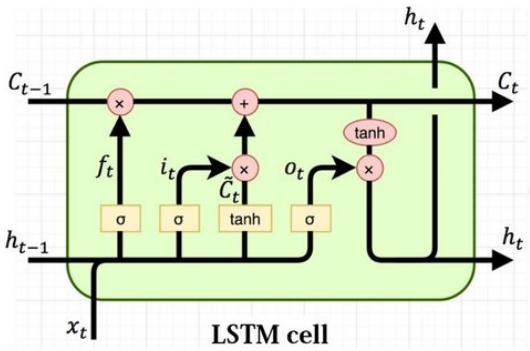
We translated the images into time series data using both the naive approach and a Hilbert curve that was built to fit our image size. We did this by mapping each pixel in the 2d-array to a one-dimensional array in the order that the chosen map dictated. The algorithm was our own. The 28×28 images were represented as a 784-step string of numbers between 0 and 1, and the 14×14 images were 196-step.

3.2 Recurrent Neural Network

To create a basic RNN, we used TensorFlow and Keras for implementation and Scikit-learn for preprocessing. We one-hot encoded the labels (i.e. $\{0, 1, ...\} \rightarrow \{[1, 0, ...], [0, 1, ...]\}$) and passed each image time-series in as a sample. The model was trained with 128 Recurrent nodes, a dropout layer with rate 0.5, a ReLU dense layer with 128 nodes, and a softmax dense output layer with 10 nodes. The loss function was cross-entropy and an adam optimizer was used. The primary metric being trained for was accuracy. We trained for 20 epochs on a batch size of 32.

3.3 Long Short-Term Memory (LSTM) Network

When training a simple RNN, we encountered memory problems. These networks have both "long-term memory" and "short-term memory", with the former being the changing weights of the network itself and the latter being certain states held by the network. While, theoretically, the short-term memory of a RNN is unbounded, in practice it is much more limited: backpropagation results in the gradient decaying to near-zero or zero in a "small" number of steps, eliminating the ability of the model to use information from too long ago in the input [6].



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 3: Visual Understanding and Equations of LSTM RNN [9]

LSTM, or Long Short-Term Memory, networks are a modified version of RNNs that have a much more robust memory. Each cell has an internal state, called the cell state C, which acts as its "long-term memory" and a hidden state h which acts as its "short-term memory". Within the cell, there are a number of gates: the "forget gate" f, "input gate" i, and output gate o, which all regulate the flow of information into and out of the system. The forget gate, naturally, determines how much information on each timestep to keep and eliminate using a weighted sigmoid function σ. The input gate determines how much information to add to the cell state, and the output gate determines how much cell state information to use in the output. This allows for much more regulation on memory, which increases the memory of these networks substantially [6].

To create a LSTM RNN, we used TensorFlow and Keras for implementation and Scikit-learn for the same preprocessing as before. The model was trained with 128 LSTM nodes, a dropout layer with rate 0.5, a ReLU dense layer with 128 nodes, and a softmax dense output layer with 10 nodes. The loss function was cross-entropy and an adam optimizer was again used. The primary metric being trained for was, again, accuracy. We trained for 20 epochs on a batch size of 32.

3.4 Baseline

For the baseline, we simply trained the image inputs on a CNN with a Conv2D ReLU layer, a MaxPooling2D layer, a Flatten layer, and two Dense layers of size 128 and 10. This was also created in TensorFlow and Keras with the same preprocessing.

4 Results

4.1 Baseline Results

Using the conventional approach, on the 28×28 image dataset, a testing accuracy of 97.28% was achieved. This is as expected, and will serve as the standard for evaluating our alternative methods. This is a very foundational baseline test, as this test itself is one of the most common neural network examples for students. The fact that this dataset/approach is so common is one of the reasons we chose this problem: it is an approach that is very successful and common, so we thought this made it a great problem to attempt to solve a different way.

4.2 Simple RNN Results

For both the 28 and 14×14 datasets with either the Hilbert or naive dimension reduction approach, accuracy results were barely above random getting consistently stuck at %10.50 and never rising higher. We believe that this is because traditional RNNs cannot effectively remember information that appeared too long ago due to the decaying gradient during backpropagation [6]. LSTM networks are a solution to this problem, so we tested them next.

4.3 LSTM RNN Results

4.3.1 Hilbert Results

We were not able to achieve any accuracy better than random on the 28×28 dataset so were limited to the 14×14 case. After processing the image data using Hilbert curves and training using on a LSTM network, we achieved a

92% accuracy on the test data. Figure 4 below shows the progression of the accuracy throughout the training epochs.

Figure 5 shows the confusion matrix of the test data. A confusion matrix provides a visual "heat map" for the digits that were often misclassified, and what the common mistakes were. As we can see, the most common incorrect labeling was predicting a 9 as being a 4, which makes intuitive sense given their visual similarity.

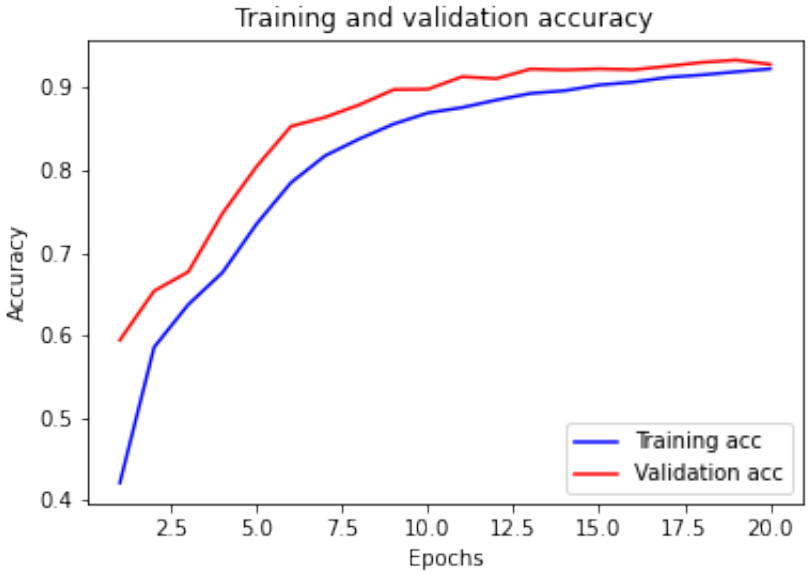


Figure 4: Accuracy over Epochs for Training and Validity sets

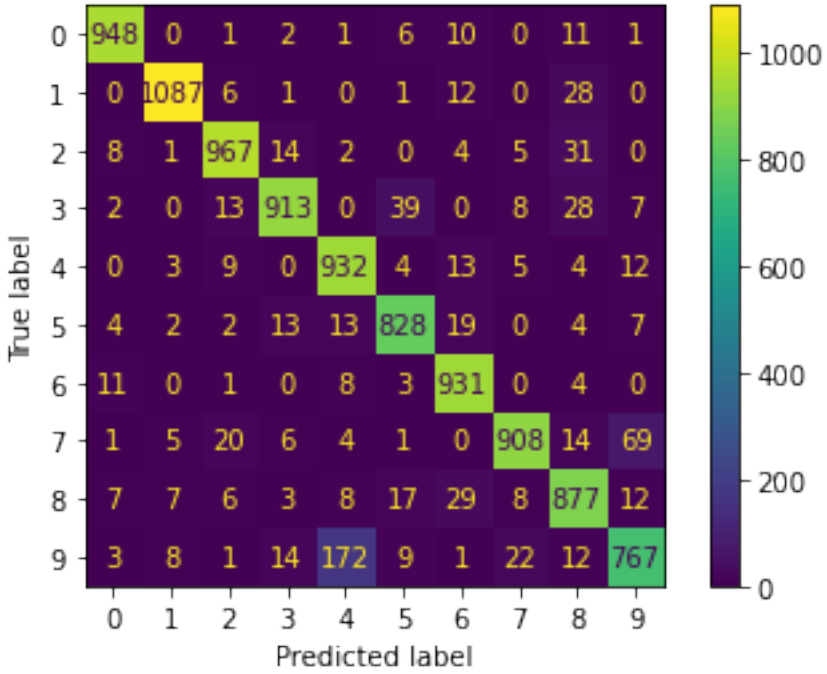


Figure 5: Hilbert Curve RNN Confusion Matrix

4.3.2 Naive Results

We were not able to achieve any accuracy better than random on the 28×28 dataset so were limited to the 14×14 case. After getting an accuracy we were happy with using Hilbert Curves, we investigated the difference between

using Hilbert curves and taking a naive approach, simply flattening the images into one-dimensional arrays row by row. We kept all other pieces of the process (neural network, image compression) the same for consistency. Interestingly, this approach worked slightly better than the previous version. It resulted in an accuracy of 97%.

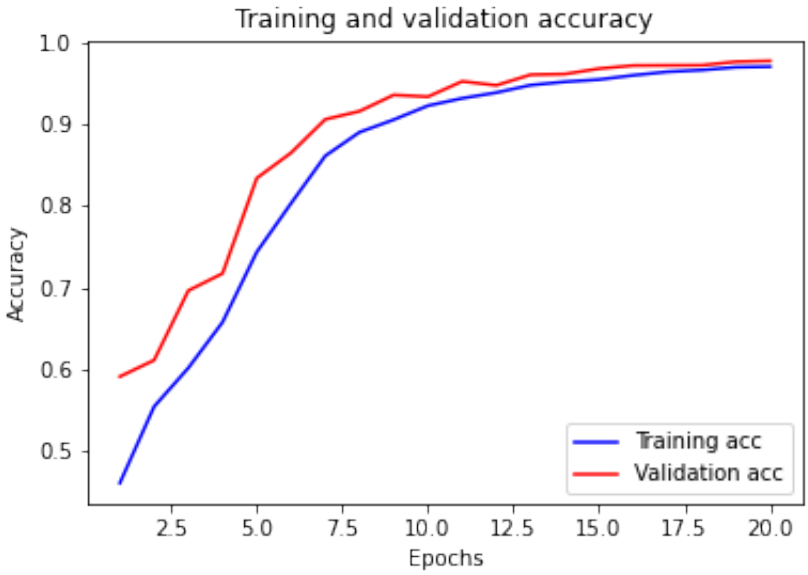


Figure 6: Naive Accuracy over Epochs for Training and Validity sets

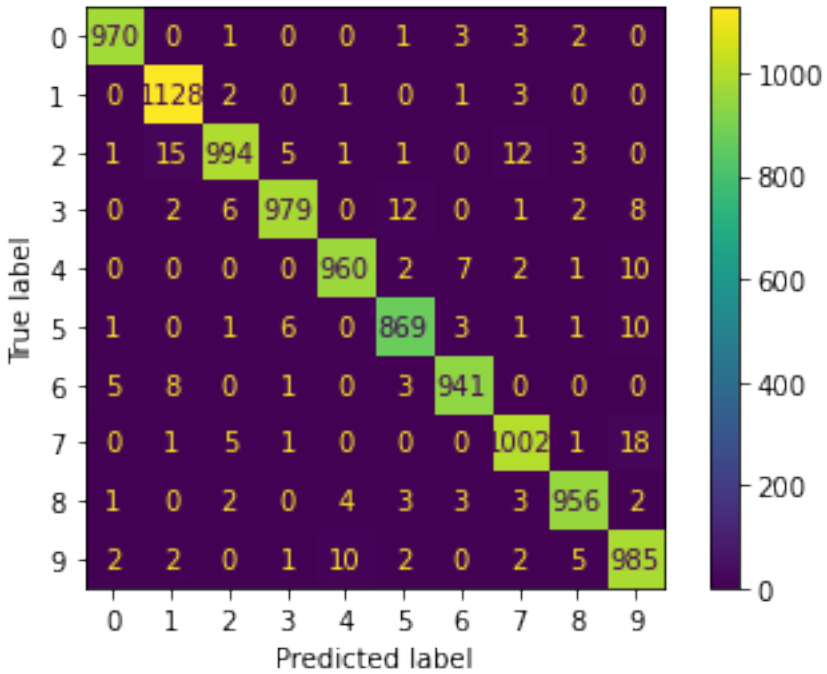


Figure 7: Naive RNN Confusion Matrix

5 Discussion

5.1 Network Memory

The most interesting result in this experiment was the powerful effect of robust network memory on RNN performance. In neither case was the less-memory-intensive simple RNN network able to capture the temporal dynamics of the image time series at either size. The LSTM RNNs, however, were able to capture the dynamics in the 14×14 case quite well for both dimension-reduction methods. This is likely due to LSTM having a more explicit method for controlling both memory and the forgetting process, but also due to LSTM networks forgetting previous information at an exponential rate (Figure 8) [4]. This means that almost all useful information from more than 150 timesteps prior is gone, meaning a 196-step image translation can fit but a 784-step image translation cannot. The information content of, say, natural language only itself decays at a power-law rate, which is likely also the case for our translation (if the value decays at all given the nature of the input).

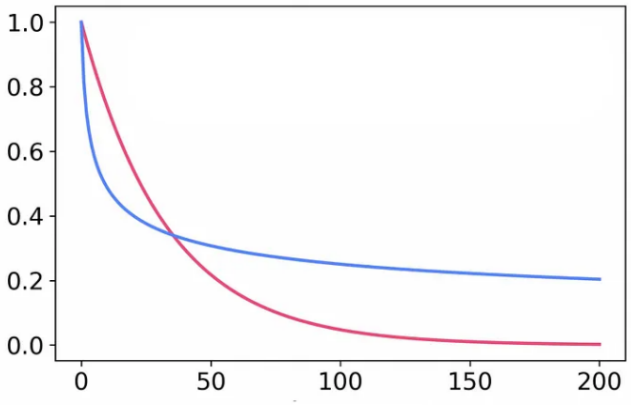


Figure 8: Exponential vs Power-law Information Loss

This is the reason for the dimension reduction being necessary. In order to fit the entire 784-step series into the memory of the network, it might be useful to attempt to implement a forget gate that forgets according to a power-law instead of exponentially. This has been shown to increase performance on tasks requiring greater than 1000-timestep network memory [4]. This was beyond the scope of our project.

5.2 Hilbert vs Naive Dimension Reduction

The next thing we discovered was that a non-naive approach to dimension reduction, the Hilbert map, underperformed the naive method. This could potentially be because the average distance of pixels containing information (non-black pixels) from the end of the time-series (where the "decision" is made) was larger for the Hilbert map than the Naive map. This means that essential information was less heavily-weighted due to the exponential decay of information.

It is also possible that the greater smoothness of the naive approach (with a periodic bit of zeros in between each line due to the edge of the image) made it easier for the network to learn the dynamics of the process (Figure 9).

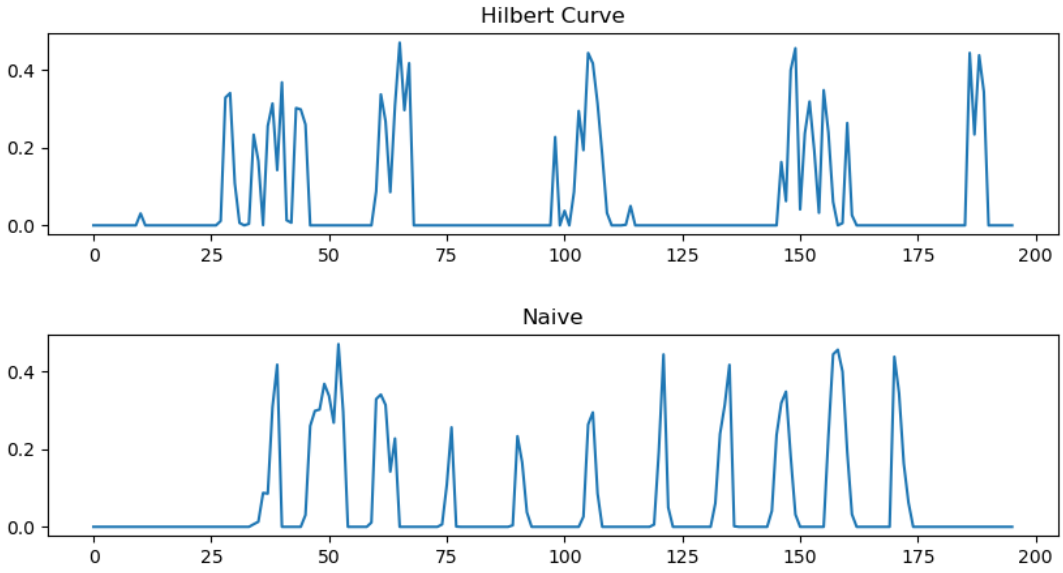


Figure 9: Hilbert Curve v Naive Time Series For Number 5

Potentially, this could lead to more optimal maps. Images of numbers tend to have their most useful pixels concentrated in the center of the image. Therefore, a map that extracts the pixels in a spiral pattern, starting on the outside and going in, could theoretically be the most effective if our conclusions are true. Testing this is also beyond the scope of this project.

5.3 Process Discussion

We found that the Hilbert Curve transformation process takes a very long time. Originally, we had a stretch goal to extend this classification technique to more complex datasets, but we did not have the computing power to process datasets larger than the MNIST digits dataset.

The biggest problem we had in this problem was getting around the limited feature set of TensorFlow RNN libraries. They do not allow control over the design of the forget gates. What was happening under the hood of the RNN is that as the neural net was processing an image's data, by the time it got to the end of the time-series type data, the majority of the data for the same corresponding image earlier had already been forgotten and was not being stored in the internal Long Short Term Memory. Therefore, it was trying to predict digits based on only a fraction of the actual image data. To solve this problem, we had to compress the images from $28 \times 28 = 784$ data points to $14 \times 14 = 196$ data points. This way, the neural net was making predictions based on all data that it should have been.

5.4 Impact

We found that, at least for this toy dataset, analyzing image data as time series can be just as accurate as Convolutional Neural Network approaches. This is particularly useful because, recently, a new form of machine learning called Reservoir Computing has emerged where the internal dynamics of a random graph of transistors can be exploited to predict the behavior of time series, particularly choatic time series like the lorenz system.

These systems are extremely simple and low-energy, but they do not generalize to non-time-series inputs. This project proves that visual 2-d inputs (and higher-dimensional non-temporal inputs as the Hilbert curve can trace through any finite-dimensional space) can be converted into appropriate time series to use this technology.

5.5 Future Plans

If we had more time and resources to work on this, we would experiment with trying this approach with more important cross-sectional data, like general image classification, handwritten English, and possibly even non-image data.

We would also test implementations of the power-law-forget-gate LSTM network to see how it fairs on the 28×28 dataset. Attempting the spiral map dimension reduction would also be an interesting experiment.

Finally, we would attempt to feed both the naive and Hilbert-generated time-series into a reservoir computer and attempt to train it on this input to see how far this generalization of time-series ML approaches actually goes.

References

[1] Mahmoud M. Abu Ghosh and Ashraf Y. Maghari. A comparative study on handwriting digit recognition using neural networks. In *2017 International Conference on Promising Electronic Technologies (ICPET)*, pages 77–81, 2017.

[2] Hue-Ling Chen and Ye-In Chang. All-nearest-neighbors finding based on the hilbert curve. *Expert Systems with Applications*, 38(6):7462–7475, 2011.

[3] Hue-Ling Chen and Ye-In Chang. All-nearest-neighbors finding based on the hilbert curve. *Expert Systems with Applications*, 38(6):7462–7475, 2011.

[4] Hsiang-Yun Sherry Chien, Javier S. Turek, Nicole Beckage, Vy A. Vo, Christopher J. Honey, and Theodore L. Willke. Slower is better: Revisiting the forgetting mechanism in LSTM for slower information decay. *CoRR*, abs/2105.05944, 2021.

[5] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 721–724. IEEE, 2017.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Ahmed Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, 3(2):334–340, 2018.

[8] Keras Team. Keras documentation: Mnist digits classification dataset.

[9] Savvas Varsamopoulos, Koen Bertels, and Carmen Almudever. Designing neural network based decoders for surface codes, 11 2018.