Optimization and Scalability of Weightless Neural Networks: A Comprehensive Study

Artem Grigor, Georg Wiese

September 29, 2023

Abstract

This report dives into the exploration and development of optimization techniques aimed at enhancing the performance and scalability of Weightless Neural Networks (WNNs). Through a thorough investigation, various strategies including data preprocessing, feature extraction, and novel model construction methodologies have been proposed and evaluated. The analyses and findings presented herein offer insight into the potential pathways toward achieving more efficient and scalable WNN architectures, opening up opportunities for their wider application in machine learning tasks.

Contents

1 Optimising the Prover

The architecture of the Weightless Neural Networks (WNN) employed in the Rust Prover has been optimized to improve efficiency without sacrificing the security integrity of the system. Several steps were considered in this optimization process, including altering the hash functions, removing redundant proofs, and applying lookup compression. Below, we discuss each of considered steps in detail.

1.1 Exploring Alternate Hash Functions

Initially, the MishMash hashing algorithm was designed with a hash function derived from raising the input to the power of three within a small finite field, and subsequently truncating the result to the desired length. This design was primarily chosen due to its ease of implementation and low computational cost. However, it lacked in cryptographic strength and seemed suboptimal for circuit settings. This section dives into the alternative hash function designs we considered.

1.1.1 Requirements

The criteria outlined for our hash function were as follows:

- High degree of randomness to ensure that similar inputs do not produce similar outputs.
- Computational efficiency.
- Straightforward verification.

The original MishMash hash function fulfilled the criteria of being quick and random. Verification was also fairly simple, necessitating only a three-degree gate, apart from the smaller finite field constraints.

However, the objective was to find a hash function that is even simpler to verify, possesses greater randomness, and is thus more secure.

1.1.2 Design

A simple multiplicative inverse was considered as an alternative hash function, as its verification was anticipated to be simple, requiring just a single multiplication gate.

The main concern revolved around the capability of the resulting hash function to sufficiently randomize the input bits.

1.1.3 Evaluation

Replacing the MishMash hash function with the multiplicative inverse, we evaluated its performance on Small, Medium, and Large MNIST datasets.

The evaluation revealed a minor performance reduction, but the inverse was found to be sufficiently random as a hash function. However, the small reduction in gate count—from a three-degree to a two-degree gate—did not justify this alteration.

1.1.4 Further Work

Aiming to further enhance the hash function's randomness, the next step involved removing the constraints for smaller finite field emulation by employing the inverse on the full field. This adjustment was expected to eliminate another four gates, reducing the total gate count to two.

This venture encountered a technical barrier: the Cuda library, used for parallel computation, supported operations only up to 64-bit integers, whereas our task necessitated handling 256-bit integers. Two potential solutions were identified:

- Transitioning from Cuda to an alternative parallelization framework that supports 256-bit integers.
- Developing a custom wrapper for Cuda to enable operations on 256-bit integers.

Although both solutions were viable, they required a significant amount of effort. We concluded that the expected benefits did not justify such an investment, particularly considering that hashing was responsible for a small portion of the total circuit size, limiting the scope for improvement.

1.2 Eliminating Unnecessary Proof Verifications

Within the existing framework, various layers of the WNN undergo proof verification. However, it was observed that verifying proofs at certain layers does not enhance the security posture of the system. A common trait among these layers is the execution of an easily invertible function f on an input value in, resulting in an output $out = f(in)$, with the original input in no longer being referenced in subsequent layers of the WNN.

The rationale behind the redundancy of proof verification in these layers can be better understood by considering two constraint systems: one with the $out == f(in)$ check (System A), and one without it (System B). The goal is to demonstrate that these systems are equivalent concerning the set of accepted assignments to out. By "equivalent," we imply that the set of accepted assignments to out remains consistent, irrespective of whether the $out == f(in)$ check is performed.

Suppose an additional assignment out_x is accepted by System B but not by System A. Given the easily invertible nature of function f, an in_x can be determined such that $f(in_x) = out_x$. Since all other checks in the systems remain unchanged, it follows that out_x is also an accepted assignment for System A, thereby affirming the equivalence of the two systems.

This equivalence stands valid only when no constraints are placed on the input value in. A typical scenario warranting input constraints occurs when the aim is to prove that a particular input in is identified as a Cat by the WNN model.

Nonetheless, there might be additional constraints on the input, necessitating proof verification for all function transitions. For example, in a competition to find an input identified as a Cat by the WNN model, the initial participant commits to their input. Once the competition concludes, they reveal the input, validate the WNN model's output, and ensure that the input hash matches the committed value. In such instances, skipping the proof verification is infeasible. Therefore, enabling this optimization should be user-configurable.

Applicable layers for this optimization include:

- Thermometer Encoding layer
- Permutation layer

It's important to note that the entire WNN model, strictly speaking, is invertible due to the lack of cryptographically secure hash functions. While inverting might be computationally intensive, it's not deemed a major concern within the context of the Rust Prover.

1.3 Compressing Multiple Values into Native Field Elements

In the operational milieu of the Rust Prover, computations are conducted in a field nearing 2^{256} in size, though the largest numbers used rarely exceed 2^{13}. This significant magnitude disparity offers an optimization avenue, particularly concerning lookups and certain sequential operations. Rather than allocating a single field element to represent a single bit or number, the approach of compressing multiple values into a single field element was adopted. This method enables a more compact representation and efficient processing.

Consider, for instance, a situation where all filter inputs must be compressed into a single field element before executing a hash function. Traditionally, this task would demand a bit decomposition and recomposition process, which, although standard, incurs additional computational overhead. By initially encoding multiple values into a single field element, the necessity for bit decomposition and recomposition is obviated, thereby saving some constraints and time.

Although the impact of this optimization might seem minor when viewed in isolation, its utility is amplified when applied to lookups, a topic explored in the ensuing section. Utilizing this condensed representation, the expansive nature of our operational field is leveraged to execute operations on encoded values more efficiently. This not only boosts the performance of the Rust Prover but also potentially sets the stage for further optimizations in managing the WNN's constraint systems, as discussed in the section on eliminating unnecessary proof verifications in Section 1.

1.4 Optimising Lookups

Lookups account for a substantial portion of the constraints, approximately 70%, making their optimization vital for enhancing the efficiency of the Rust Prover.

1.4.1 Folding

Utilising a folding scheme presents a viable strategy for optimising lookups. Recent developments in folding schemes like Sangria now support the folding of custom PLONK gates, which can be utilized for implementing lookups. Additionally, frameworks like Origami provide a direct blueprint for implementing lookups in HALO2. Although these schemes haven't been extensively audited, their potential to considerably reduce constraints is promising. Given that the WNN involves numerous repeated lookups and operations, such as each discriminator performs identical operations just with different inputs, the reduction in constraints could be significant.

However, to the best of our knowledge, Origami hasn't been implemented, and Sangria remains inaccessible for direct usage in the Halo2 library. We are hopeful that this situation will improve in the near future.

1.4.2 Compression

Lasso A recent innovation, Lasso, has illuminated the path for lookup optimizations. Lasso requires payment only for the lookups actually performed, while the rest of the lookup table is accessed freely. Given that in WNN, on average, only 2 to 6 values out of 2^{13} are looked up, the reduction in constraints could be significant, potentially accelerating operations by a factor of 1000, depending on the intrinsic costs of the scheme.

However, Lasso, being newly introduced, is not yet compatible with the Halo2 library.

Custom Compression To refine the lookup process further, we endeavoured to develop an efficient protocol capable of provably decoding the i^{th} bit of n bits encoded in a single field element within finite field arithmetic. This protocol leverages the unique characteristic of numbers within a finite field, where some have square roots while others do not, to facilitate the encoding of bits.

Our protocol aims to compact n bits into a single field element and to provide a proof for the value of any given i^{th} bit among the encoded n bits, remarkably within a single constraint. This scheme emerged from the need to compress binary lookup tables with sparse read access and accelerate bit decomposition, significantly reducing the row size of such tables by a factor of n with only a single additional constraint per looked-up value, by compressing multiple consecutive evaluations into a single field value.

To our knowledge, this protocol is the first of its kind and is currently under review.

At the heart of the protocol is a pre-computation phase dedicated to constructing an encoding table. Once created, this table serves as a universal asset, negating any further recalculations. Post this pre-computation, the encoding of n bits becomes a straightforward task, leveraging the table. For instance, encoding bits '[1, 0, 1]' requires identifying a number x that adheres to a set of conditions based on the square root property of x or x offset by specific values—a process significantly expedited by the precomputed encoding table.

The decoding operation of the i^{th} bit involves examining specific sums involving the encoding field element enc_v and $2 \times i$ to determine whether they are squares, which in turn reveals the bit value. The proof of the bit value utilizes the verifiability of the square root property: by providing the square root of a particular expression, it becomes possible to corroborate the i^{th} bit value with a singular constraint. As a result, the full decoding of all n bits demands merely n constraints, making this protocol a practical extension to our Lookup Compression scheme.

Regarding computational complexity, the pre-processing stage crucial for constructing the encoding table bears a time complexity of $O(4^n)$ and a space complexity of $O(2^n)$. However, the encoding and decoding processes are significantly less computationally intensive, rendering this scheme a plausible candidate for augmenting the efficiency of the lookup process in our architecture. We further expect that more efficient algorithms for constructing the encoding table will be discovered in the future, thereby reducing the time and space complexities of the pre-processing stage.

Earlier Approach: 3-Level Array Lookup Before exploring our custom compression scheme, we utilized a 3-level array lookup mechanism to optimize extraction of bit compressed with other bits and stored within finite field elements. Recognising that a single field element could encapsulate up to M bits of information, we aimed to encode multiple index bits into one field element initially using simple concatenation.

The procedure bundled n indices together, with $n < M$. During lookups, we first determined a compression index $comp\_index = index_i/n$, signifying a compression of n bits. The compression at this stage was a concatenation $f_1..f_n$, resulting in a number between 0 and 2^n.

To ascertain the value of the i-th bit stored in the compression, several strategies were explored:

1. **Decomposition:**
   - Formulate $pre\_part + flag_i \cdot 2^i + post\_part = f_1..f_n$,
   - Range check $0 \leq pre\_part < 2^i$,
   - Range check $2^i < post\_part < 2^n$,

   necessitating 2^n lookup rows with 2 columns.
2. **Index Combination Lookup:** A lookup table encompassing all possible combinations between n bit values and a bit index was created, enabling the direct extraction of the i-th bit, requiring $2^{(n \cdot n)}$ lookup rows with 3 columns.
3. **Chunk Decomposition:** This strategy converted the problem into selecting a correct chunk of k bits from n bits, with the potential for recursive application.
   - Establish $chunk_0 = acc_0$,
   - $chunk_i + acc_{i-1} = acc_i$,
   - $select\_chunk_j = select\_chunk_{j-1} + chunk_j \cdot (j == select_i)$,

   necessitating $2^{(n-k)}$ rows and approximately 7 columns.

The original compression schema proposed and employed in the previous milestone was a hybrid of strategies 2 and 3. An optimization equation has been devised to evaluate the optimal chunk sizes, their quantity, and the point at which an index lookup should be executed.

For instance, in the previous milestone, we used Chunk Decomposition with $k = 3$, and $n = 5$, followed by an Index Combination Lookup where $n = 3$,. This necessitated $2^9 + 2^2$ rows.

However, this solution displayed scalability issues. There existed a threshold beyond which compression yielded diminishing returns. Eventually, the cumulative cost of index combination lookups and chunk decompositions would surpass the cost of straightforward lookups.

Although the exact figures were elusive and necessitated solving the optimization equation to ascertain optimal chunk sizes and their count, the emergence of a scheme capable of compressing n bits into a single field element without escalating overhead prompted us to set aside the optimization dilemma. Instead, our focus shifted towards refining the newly proposed compression scheme, which offered a more promising avenue for enhancing lookup efficiency and scalability.

2 Optimising Preprocessing in WNN

Data preprocessing and feature extraction are familiar steps in traditional machine learning, with a track record of boosting model performance in many experiments. Yet, their effectiveness remains largely untested in the realm of Weightless Neural Networks (WNNs). The assumption that standard preprocessing and feature extraction techniques would work equally well with binary data and models is quite looming, especially given that most of these techniques are tailored for operations with floating-point numbers.

In the sections that follow, we present methods aimed at enhancing the performance of the cutting-edge WNN model BTHWeN.

2.1 Data preprocessing

Data Augmentation

Data augmentation is our first tactic to combat overfitting and to better equip the model for generalization by enlarging the training dataset with minor variations in the data. Two conventional methods were employed.

Initially, we applied random shifts to the images. For each data point in the training set, two values were selected at random: a horizontal shift and a vertical shift, with negative values representing shifts to the left or downward, respectively.

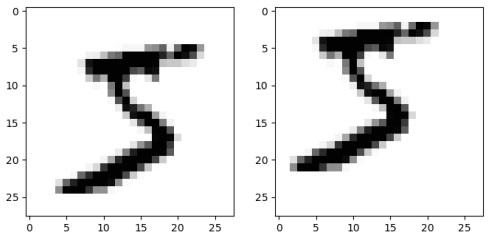

Figure 1: Data augmentation: image shifting

A common approach in traditional machine learning is to add noise to the data, but this isn't a viable strategy for WNNs due to the necessity of binarizing the data. Any noise added prior would be lost post-binarization, leading to identical binary data. Hence, we suggest an alternative technique.

Post thermometer encoding, a specified number of bits are inverted. Blindly inverting bits might yield unrealistic images and potentially hinder the model (e.g., altering central pixels inside the digit 0). Therefore, we propose a defined method: initially, a probability p is set. Following this, every black pixel with at least one white pixel neighbor can turn white with probability p, and vice versa. This process, while altering the image, retains its recognizability. With the MNIST dataset, the numbers remain identifiable, albeit in varying handwriting styles (see 2).

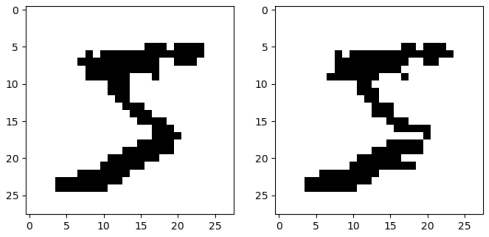

Figure 2: Data augmentation: image corruption

It's notable that data augmentation is particularly beneficial given the learning mechanism of

WNNs, which essentially memorize each data point provided. Hence, during inference, a slight shift in the image could stump the model. A common workaround is to partition the image into pixel groups randomly, although this doesn't always yield the desired outcome as it heavily relies on the randomness of the partitioning. Data augmentation can mitigate this randomness by diversifying the training data.

The data augmentation process doesn't impact the model's size or its processing speed during inference, although, as expected, it prolongs the training duration due to the enlarged dataset.

Discarding Non-informative Bits

Post binarization, especially with thermometer encoding, certain non-informative bits emerge. These bits are consistent (or nearly so) across all dataset observations (e.g., corner pixels being 0 for all images). Evidently, such bits offer no insight regarding the class an observation belongs to. We label these bits as "bad" and exclude them henceforth. Therefore, when the image is divided into bit groups, these "bad" bits are left out.

Dimensionality Reduction

In certain scenarios, there might be a need to reduce the data's dimensionality, for instance, when handling high-resolution images. We are in pursuit of a relatively swift method that effectively shrinks the observation size. Pooling (min, max, avg) emerges as such a method. Given that the data is binarized anyway, employing this method doesn't entail a significant loss of information, provided the filter size is relatively small compared to the initial image. Interestingly, some images might actually benefit from pooling, as it diminishes blurriness near color transition edges, which is favorable for binarization.

2.2 Feature Extraction

In any multilayer neural network, feature extraction is a fundamental capability, often harnessed in knowledge transfer. We aimed to adopt the beneficial practices from traditional neural networks and adapt them for use in weightless neural networks (WNNs). The goal was to enable the model to autonomously discern beneficial features from detrimental ones. However, the challenge is, how can this be achieved without the luxury of classical model training? Backpropagation, while a potent tool for conventional neural networks, renders models unexplainable. Unlike most models, in WNNs, we can meticulously evaluate each learned feature and discard it if found to be redundant.

Feature Extraction Strategy

Initially, it's crucial to understand what constitutes a feature in the context of a WNN. While a single bit post-binarisation could technically be considered a feature, it often lacks substantial information on its own. Henceforth, we'll define a feature as a cluster of bits routed to a specific filter. In standard WNNs and BTHOWeN[1], an observation is manually divided into bit groups, with each group assigned to a particular filter. This setup allows us to identify which filter corresponds to which feature. By assessing the performance of each filter independently, we can gauge the relevance of each feature. The remaining task is to establish a measure of feature significance.

To address this, we need to clarify what we expect from an individual feature. It's unreasonable to expect a single feature to accurately classify an observation. A situation where a filter returns **1** only in the discriminator aligned with the class of the current observation is extremely unlikely, because a single feature could be common to multiple classes. For instance, both digits "6" and "8" have a circular bottom, making "having a circular bottom" a useful feature as it helps the model instantly segregate these two digits from others using just this feature. However, a feature like "empty space in the image corner," despite returning **1** in the correct class discriminator, would also return **1** in all other discriminators, rendering it ineffective. In essence, we want a filter representing a feature to consistently return **true** for its class, while returning **false** for at least some other classes.

To rigorously determine feature significance, we introduce two metrics. Let's denote the set of features as $\mathcal{F} = \{f_k | k = 1, \ldots, M\}$ and each filter within a model as ϕ_{j,f_k}, where j is the number

[1]The WNN structure in focus

of discriminators (or number of classes) and f_k is the feature corresponding to the given filter. We define the accuracy of a feature as the ratio of occurrences when the filter associated with the feature in question returned **1** in the discriminator corresponding to the class of the considered observation (1) to the total number of observations.

$$acc(f_k) = \frac{\sum_{i=1}^{N} \mathbf{1}_{\phi_{y_i,f_k}(x_i)=1}(x_i)}{N} \tag{1}$$

where $X = \{x_i | i = 1, \ldots, N\}, Y = \{y_i, i = 1, \ldots, N\}$ represent observations and labels respectively. Next, we define the commonality of a feature as the ratio of discriminators for which the corresponding filter returned **1** to the total number of discriminators (number of classes) (2), averaged over all observations. A value of $ord(f_k) \approx 1$ indicates a feature akin to "empty space in the image corner" described earlier.

$$ord(f_k) = \frac{1}{N} \sum_{i=1}^{N} \frac{\sum_{j=1}^{C} \mathbf{1}_{\phi_{j,f_k}(x_i)=1}(x_i)}{C} \tag{2}$$

where C denotes the number of discriminators (number of classes).

Subsequently, we introduce two hyperparameters $\alpha \in [0, 1], \beta \in [0, 1]$. A feature is deemed α, β-significant if:

$$(acc(f_k) \geq \alpha) \wedge (ord(f_k) \leq \beta) \tag{3}$$

and the set of $\alpha\beta$-significant features is represented as $\mathcal{F}^{\alpha\beta} = \{f_k | (acc(f_k) \geq \alpha) \wedge (ord(f_k) \leq \beta)\}$. Suitable values for α, β can be determined either through logical deduction (given their intuitive interpretation) or via a validation set.

With $\alpha\beta$-significance defined, we can now utilize this criterion for feature selection. We suggest two applications for feature selection in WNNs: model reduction and model construction.

Model Reduction via Feature Selection

Model reduction, based on $\alpha\beta$-significance, entails the removal of filters associated with $\alpha\beta$-insignificant features. This method allows for model size reduction by discarding less important features.

Model Construction using a Greedy Algorithm for Feature Selection

Typically, features for the BTHOWeN model (or other WNNs) are chosen through a random split of the binarised image into pixel groups. This random approach doesn't ensure the inclusion of all useful features. To build a model comprising important features, we propose the following greedy algorithm:

1. Initialize $\mathcal{F}^{\alpha\beta} = \{\}$. Define hyperparameters α, β and the number of $\alpha\beta$-significant features to be included, denoted as γ.
2. Generate $\mathcal{F}$ by randomly splitting the binarised image into pixel groups.
3. For each f_k in $\mathcal{F}$:
   (a) Train a model with a single feature.
   (b) On the validation set, find the bleach value that ensures $ord(f_k) \leq \beta$ while maximizing $acc(f_k)$.
   (c) If $acc_{\max}(f_k) \geq \alpha$ and f_k doesn't share more than half of its pixels with any $f^{\alpha\beta} \in \mathcal{F}^{\alpha\beta}$ (to avoid similar features), add f_k to set $\mathcal{F}^{\alpha\beta}$ and save its optimal bleach value.
   (d) If $|\mathcal{F}^{\alpha\beta}| = \gamma$, halt the process.
4. If $|\mathcal{F}^{\alpha\beta}| < \gamma$, revert to step 2.

Note, this algorithm not only facilitates the creation of a set of $\alpha\beta$-significant features but also approximates the best bleaching value for each feature, enabling the model to learn more sophisticated patterns.

2.3 Evaluation of Data Preprocessing and Feature Selection

The methods of data preprocessing and feature selection laid out earlier can be utilized in various combinations. The suitability and effectiveness of each method should be gauged individually for each task. Additionally, the success of a preprocessing method is also influenced by the model architecture. Below are some conclusions drawn from simulations:

2.3.1 Outcomes on Data Augmentation

Data augmentation can enhance the model's performance, but necessitates a larger WNN model since WNN learns by recognizing certain patterns. Data augmentation creates data with a wider variety of patterns. Hence, a larger model is needed to identify all possible patterns, otherwise, we encounter an issue where the hashing function begins to map different patterns to identical values. Thus, when employing data augmentation with small models, practitioners should exercise caution as it might degrade the performance (see table 1).

model	accuracy
Small MNIST	0.924
Small MNIST with data augmentation	0.913
Large MNIST	0.945
Large MNIST with data augmentation	0.948

Table 1: Comparing models on MNIST dataset with and without data augmentation. Small model specifications: bits per input = 2, unit inputs = 28, unit entries = 1024, unit hashes = 2, Large model specifications: bits per input = 6, unit inputs = 49, unit entries = 8192, unit hashes = 4

For smaller models, applying pooling may be beneficial. As mentioned earlier, small models struggle to learn a variety of different patterns. Pooling reduces the size of the observations while preserving important patterns (see table 2). Note, for the same models, the model applied to data with pooling with window 2×2 has half the filters, thus is half the size. As a result, we have a model that is half as small, but its accuracy is only 1% lower.

model	accuracy
Tiny MNIST	0.887
Tiny MNIST, max pooling with window 2 by 2	.876

Table 2: Tiny model on MNIST dataset with and without pooling. Tiny model specifications: bits per input = 2, unit inputs = 9, unit entries = 128, unit hashes = 1

2.3.2 Outcomes on Model Reduction using Feature Selection

The experiments conducted reveal that the proposed method enables effective model size reduction by removing filters linked to $\alpha\beta$-insignificant features. The table shows that even for a small model, we were able to cut the model size by half (removing 24 features out of 56), with only a 1.3 percent drop in accuracy.

model	accuracy
Small MNIST	0.924
Small MNIST with feature selection	0.911

Table 3: Small model on MNIST dataset with and without features selection. Small model specifications: bits per input = 2, unit inputs = 28, unit entries = 1024, unit hashes = 2

2.3.3 Outcomes on Model Construction using Feature Selection

The greedy algorithm detailed earlier is computationally demanding, which notably extends the model's training time. Based on the experiments, it is concluded that for simpler datasets, applying this

algorithm isn't reasonable in most cases if model parameters are properly adjusted, as it offers minimal accuracy improvement. For instance, on the MNIST dataset, if we set `unit_entries`[2] greater than 28 (number of bits counted as one feature), the likelihood of any feature being insignificant is very low. Consequently, the model built by the proposed greedy algorithm exhibits nearly the same performance (see table). Nonetheless, for larger and more complex datasets, this method is anticipated to yield better results.

2.3.4 Merging Approaches

For attaining a model with superior performance, a blend of the proposed methods can be employed. The procedure that aided in achieving a better model on the MNIST dataset, compared to the BTHOWeN model of a similar size, is described below:

1. Apply max pooling to the images with a kernel of size 2×2.
2. Apply thermometer encoding with bits per input = 2
3. Identify uninformative bits and label them as "bad"
4. Generate features. Rather than creating random features, a sliding window of size 3×3 is defined and placed in each possible position. If no bit is "bad", this combination of bits is added to features.
5. Following this procedure, numerous features are obtained, many of which overlap with each other. Therefore, a model is constructed using the greedy algorithm for feature selection.

For this method, the following parameters for the model were defined: bits per input = 2, unit inputs = 9, unit entries = 512, unit hashes = 2. The number of features used is 125[3]. Thus, a model of roughly the same size as the MNIST small model from the article [**?**] (MNIST small model has double the filter size, but our model has double the filters) was obtained, but with 1% better accuracy. The accuracy for the given model is equal to 0.944.

3 Scaling WNN

Training WNNs for more complex datasets having many features inevitably leads to a significant increase in the model size and, consequently, inference time [**?**]. To address this issue, we can employ certain techniques mentioned earlier.

Firstly, we can apply pooling, which significantly reduces the size of the data. However, it's important to note that some information may be lost due to this operation.

Secondly, we can remove uninformative bits. This operation doesn't eliminate any information, but it's worth noting that not every dataset contains such bits.

Thirdly, we can utilize the feature selection algorithm proposed earlier. Instead of using all the bits from the encoded observations, we can explicitly control the number of features we want to use and sample significant features using the proposed greedy algorithm. As demonstrated in section 2.3.4, this combination of methods can dramatically reduce the model's size and, with careful hyperparameter tuning, even improve accuracy.

As an illustrative example, we attempted to fit the BTHOWeN model to the CIFAR-10 dataset. Even the Large MNIST model from the article [**?**] resulted in an accuracy of only 0.43. This suggests that it may be necessary to significantly increase the size of the model, apply the aforementioned approaches, or a combination of both, to achieve satisfactory performance.

[2] hyperparameters names are carried over from the article where the BTHOWeN model is introduced

[3] Note that if data preprocessing and feature selection weren't applied, the total number of features for 2 bits per input would be $(28 - 3 + 1)^2 = 676$

4 Translating DNN into WNN

Translating DNNs into WWNs is a challenging task. Binarizing the weights is not a viable option, as rounding weights to 0 or 1 for a trained DNN effectively results in a completely new DNN. Theoretically, one approach is to extract features from a DNN, which can then be used for feature selection in a WNN. However, this approach immediately encounters the unsolved problem of creating an interpretable deep neural network.

Some attempts have been made to create interpretable neural networks by combining Formal Concept Analysis with neural networks [**?**, **?**, **?**]. However, this represents a relatively new direction in neural network research, and as of now, there are no universally satisfying results in this area.

5 Efficiently Training WNN

Traditional training methods for WNNs are not applicable, as gradient descent works only with floating-point numbers and cannot be directly applied to discrete tasks. In light of this, we can only propose a high-level concept, which is still in its early stages of exploration.

Firstly, we may attempt to formulate the task of training the WNN model as a discrete optimization problem.

Secondly, many discrete optimization tasks can be reduced to a Quadratic Unconstrained Binary Optimization (QUBO) problem, which involves minimizing a second-degree polynomial where all variables are either 0 or 1. This problem is equivalent to finding the ground state in the Ising model, where the corresponding spin variables are either -1 or 1. Essentially, we need to find a spin configuration that minimizes the Hamiltonian of the Ising model [**?**]. This problem can further be reduced to solving a system of differential equations [**?**, **?**].

Hence, we may explore the possibility of constructing a transition from a discrete (binary) problem to a continuous problem and then investigate how classical training algorithms can be adapted for use in this context.

6 Conclusion

In this report, we have discussed significant improvements in both the Rust Prover and Weightless Neural Networks (WNNs), aiming to enhance their efficiency and applicability. These advancements contribute to the broader fields of cryptography and neural network research, offering promising avenues for future developments.

6.1 Rust Prover Improvements

The Rust Prover has seen substantial improvements in optimizing lookups, a critical component accounting for a significant portion of constraints in cryptographic protocols. Two primary approaches have been explored:

6.1.1 Folding

The utilization of folding schemes like Sangria and Origami holds significant potential to reduce constraints associated with lookups in cryptographic proofs. While these schemes are promising, they require further development and integration into the Halo2 library to realize their full benefits. The community's continued efforts in this direction are crucial for advancing efficient cryptographic protocols.

6.1.2 Compression

Lasso and custom compression schemes have been introduced as innovative methods for optimizing lookups. Lasso, in particular, shows promise by drastically reducing constraints for sparsely accessed lookup tables. However, its compatibility with Halo2 needs to be addressed. The custom compression scheme, which compresses binary lookup tables into a single field element, offers exciting possibilities for improving lookup efficiency.

6.2 WNN Improvements

Improvements in data preprocessing and feature selection for Weightless Neural Networks (WNNs) have been explored to enhance their performance. Key findings include:

6.2.1 Data Augmentation

Data augmentation can be a valuable tool for combating overfitting and enhancing the generalization of WNNs. However, caution must be exercised when applying it to smaller models, as it may lead to degradation in performance. Larger models are better suited to handle the increased variety of patterns introduced by data augmentation.

6.2.2 Model Reduction using Feature Selection

The proposed feature selection algorithm has demonstrated effectiveness in reducing model size while maintaining reasonable accuracy. Even for smaller models, significant reductions in size (up to 50%) can be achieved with only a modest drop in accuracy.

6.2.3 Model Construction using Feature Selection

The greedy algorithm for feature selection, while computationally demanding, offers a means to construct models with important features. Its impact varies depending on the dataset complexity. For simpler datasets like MNIST, other model parameters may render this method less effective, but for larger and more complex datasets, it holds promise for achieving better results.

6.3 Further Work

To continue advancing the Rust Prover and WNNs, several avenues for further work are suggested:

6.3.1 Improved Lookup Compression

Efforts should focus on improving lookup compression algorithms, such as Lasso, and ensuring their compatibility with existing cryptographic libraries like Halo2. Exploring novel compression techniques that reduce constraints in lookup operations remains a critical research area.

6.3.2 Feature Selection on Larger Datasets

Extending the application of feature selection algorithms to larger and more complex datasets is imperative. Evaluating the performance and scalability of feature selection on datasets beyond MNIST will provide valuable insights into its practical utility.

In conclusion, the advancements made in both the Rust Prover and WNNs have the potential to impact their respective fields significantly. By addressing the suggested further work items, we can continue to enhance the efficiency and effectiveness of these technologies, opening up new possibilities for cryptography and neural network applications.