Final Project - Final Report

2020-2021

HIERARCHICAL TABLE ENSEMBLE (HTE)

Deep Learning Algorithm

Omri Dahan & Shira Shushan

Advisor - Dr. Aharon Bar-Hillel

Co-advisor - Ph.D student Guy Farjon

Department of Industrial Engineering and Management

Faculty of Engineering

Ben-Gurion University Of The Negev, Israel

June 2021

1 Acknowledgement

We would like to express our sincere gratitude to **Dr.Aharon Bar Hilel** who grant us the opportunity to be a part of the HTE project, guide us on the Learning, Representation, Vision course and for general Deep-Learning guidance.

We would also like to express our special appreciation to **Ph.d student Guy Farjon** for weeks and months of support, patience and dedication in effort to help us achieve success.
We had the golden opportunity to be a part of a project which have the potential to impact the global approach for Deep Learning. HTE is an innovative project which we hope to see evolve and become a new tool for academic and business problems. We are grateful to take part of it.

Contents

List of Tables

List of Figures

2 Abstract

Deep learning has emerged as an important approach to solve both academic and real-world problems. In this approach, multiple layers (neural layers or others) are used, hierarchically, to progressively extract high-level features from the input data. One common deep learning model is the deep Artificial Neural Network (ANN). Contains millions of 'neurons' which usually requires the need of GPUs or other specialized hardware. ANNs had empirical success in the last decade and currently it is one of a few alternative existing in practice. We claim that there are other alternatives for deep learning, resulting with similar results but with faster performance.

Our proposed alternative is the Hierarchical Table Ensemble (HTE) algorithm. This algorithm uses ferns as the main computational component. Ferns are like decision trees, with a small but significant difference - the queries at each level 'i' are known in advanced and are not dependent on the results of level 'j'. Each query is a simple binary question which is used for quicker calculations and optimization. These ferns are word-calculators and their output is a K-bit binary codeword which is used as an index into a 'voting table' which retrieves the corresponding output representation. By applying M such ferns, and summing their results, we get an ensemble of simple voting tables. HTE will be a stacked model of L such layers, connected one over the other to extract high level features for the input.

Other deep classifiers exist, such as "gcForest", which is a deep random forest. HTE algorithm aims to compete with such architectures, enabling deep learning framework but as a computationally cheaper approach. As a first, we conducted experiments on synthetic data to tune the classifier's hyper-parameters. Seconds, after we were able to solve synthetic problems, we moves to real-world data. We compared results on publicly available datasets such as ADULT, LETTER, IMDB, WINE etc. with the "gcForest" algorithm. Currently, "gcForest" shows slightly better results at this point. For example, on the ADULT dataset which aims to figure out if a person is likely to earn 50K$ or more a year, only by knowing that person's demographic information, the "gcForest" accuracy was 86% while HTE showed 83%.

Finally, HTE is a new deep algorithm which is still being researched. HTE suggests a framework that enables accelerated CPU inference for low-compute domains, with mild costs of accuracy descending and memory consumption. In our work we present how HTE works on synthetic examples and compare the algorithm results on various datasets with other known deep learning algorithms.

Key Words: Bit Functions, Ferns, Machine Learning, Deep Learning, Hierarchical Table Ensemble

3 Introduction

In the last few years, deep learning has emerged as an important approach to solve both academic and real-world problems. In this approach, multiple layers (neural layers or others) are used, hierarchically, to progressively extract high-level features from the input data. One common deep learning model is the deep Artificial Neural Network (ANN), which is a graph of dot-product computing elements. It is customary to think there is an essential tight coupling between these two concepts, probably because ANNs had empirical success in the last decade and currently it is one of a few alternative existing in practice. We claim that there are other alternatives for deep learning, resulting with similar results but with faster performance.

Deep ANN models contain millions of 'neurons' which usually requires the need of GPUs or other specialized hardware. In effort to make deep learning computationally affordable for common devices like drones, smartphones, security cameras, IOT devices and others CPU-based devices, a simple alternative could be Hierarchical Table Ensemble (HTE). For a CPU with ideally unbounded memory access capabilities, the fastest classifier would be based on a single huge table (hypothetically). By applying a sequence of simple binary queries to the input and build an index from the resulting bits, the answer will be retrieved from the table entry.

An alternative realistic approach will be to use transformation called 'ferns' (see fig. 1). Ferns are similar to decision trees, with a small but significant difference - the queries at each specific level i are known in advanced and are not dependent on the results of level j. Each query is actually a simple binary question which is used for quicker calculations and optimization. These ferns are word-calculators ant their output is a K-bit binary codeword which is used an index into a 'voting table' which retrieves the corresponding output representation. By applying M such ferns, and summing their results, we get an ensemble of simple voting tables. HTE will be a stacked model of L such layers, connected one over the other to extract high level features for the input.

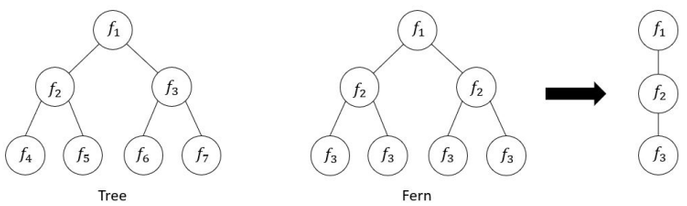

Figure 1: Ferns description: similar to decision trees but with a significant difference - in decision trees the query in each node will be chosen by some splitting criteria, and the same query won't appear in more than a single node. In a fern, the queries at each specific level i are known in advanced and are not dependent on the results of level j, where $i > j$.

In summary, we will try to present an alternative to neuron-based networks called HTE for structured data. From an applicative point of view, HTE suggests a framework enabling accelerated CPU inference for low-compute domains, with mild costs of accuracy descending and memory consumption. In this report we will present how HTE works on synthetic examples, and how HTE perform on various datasets compare with other known deep learning algorithms. Finally, we will conclude the results of our tests in effort to support that HTE can be a good alternative.

4 Related Work

4.1 Deep Learning and Convolutional Neural Networks

As stated, deep learning is a method to extract high-level features from the input data. This approach was driven from the data processing mechanisms in the human brain for usage in identifying objects, translating languages, recognizing speech, and making decisions. Currently, neural networks are the most popular deep learning models due to several factors. **First**, as the name implies, scientists and engineers were inspired by the biological neural structure of the human brain. However, current simple hardware, such as CPUs, cannot operate parallelly like biological brain neuron structures. Instead, CPUs offers the ability to accesses great memory structures efficiently. **Second**, there are some theoretical factors in favor of neural networks which are known, such as the universal approximation - neural networks has the capability to estimate each continuous function on a dense domain using only two hidden layers. **Third**, and probably the main factor, is their observed success. Nonetheless, there are claims that these factors are incontestable and that an alternative of voting tables might be better in low compute fields [1].

Deep ANNs models were planned and built by an American psychologist, Frank Rosenblatt, in 1958 [2], to let the system learn rules, recognize patterns and make decisions in a human-like fashion. It does so by a weight modification rule and their purpose is to solve a given task, which is specified by feeding the network with input-output examples. Deep ANNs are made of several ANN layers - the input layer X, the output layer Y, and several intermediate layers $\{l_i\}_{i=1}^{L}$. The intermediate layers can consist different number of neurons, that can interact with one another. The deep ANNs can be thought of as a directed graph in which connections within a layer or from an higher layer to a lower layer are prohibited. The input vector sets the input unit's state. Then the state of every neuron in each layer is determined by applying a non-linear function on the output of the previous layer.

Convolutional Neural Network (CNN) is an advanced ANN model. It was mainly intended to learn spatial hierarchies of attributes over backward expansion by means of numerous building blocks, like convolution layers, activation functions, pooling layers, ANN layers, and more. These networks can assist in originating related image data in form of minor patches or image units. The neurons in the convolution layers are responsible for the cluster of neurons in the earlier layer. The working process of CNNs comprises of four main layers

1. Convolutional layer - this layer is the core building block of CNNs and is designed to identify the features of an image. Each such layer includes several filters which are the learnable parameters of the layer. These filters are usually small, and are multiplied by the input data in an convolutional manner.

2. Activation function - this function introduce the non-linear property of the decision function and of the overall network. One of the most common activation function is the ReLU which is a element-wise activation function thresholding at zero - $ReLU(x) = max(0, x)$.

3. Pooling layer - this layer was designed to reduce the number of parameters to learn by down-sampling the input representation. One common layer is the max-pooling layer, in which the input is partitioned into sub-regions, and for each of them, only the maximum value is kept.

4. Fully-connected layer - these are standard ANN layers which can be incorporated, usually at the end of the CNN model.

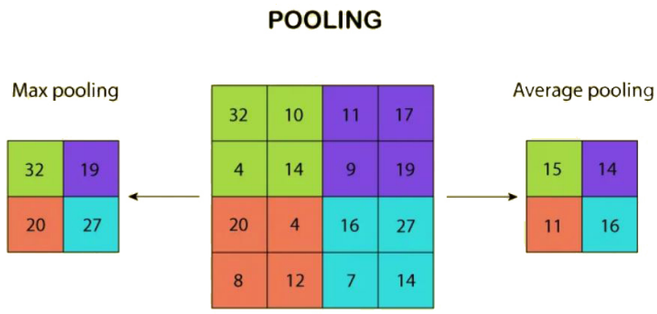

Figure 2: CNN pooling: Progressively reduces the spatial size of representation to reduce amount of parameters and computation in network and also controls overfitting. If no pooling, then the output consists of same resolution as input.

As of today, CNNs are popular and are used in an abundance of applications. However, as stated in the introduction, CNNs usually requires special hardware (like GPUs) even at inference time. There are several approaches to accelerate CNNs, for example, recently [3] offers the ability to optimize inference CNN models on CPUs. But still, as most of the acceleration approaches, the convolutional layers, which is the most computational-expensive component in CNNs, are kept. Since this hardware is expansive and not adequate for each end-device (such as smart-phones or drones) optimizing the CNN inference models on standard CPUs will present an important gain to a large number of users.

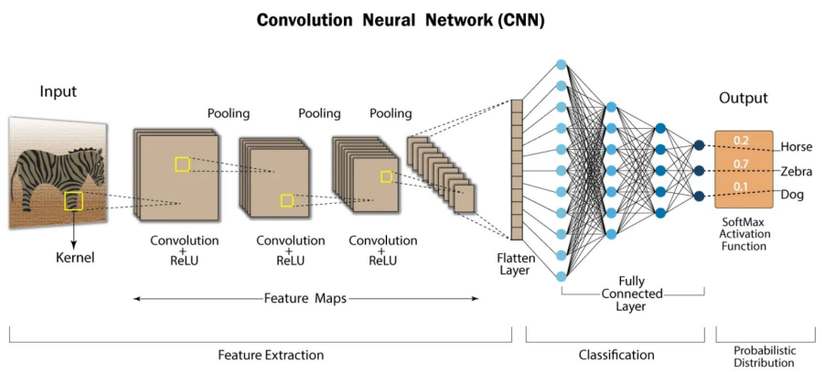

Figure 3: CNN full architecture

4.2 Convolutional Tables Ensemble (CTE) and Deep CTE

The CTE algorithm was created for image classification as a computational cheaper alternative for CNNs. The CT transformation is build as a pair of functions (W, V) where W is a word calculator and V is a voting table. A single neuron of the word calculator W computers the answer of K simple binary questions over the input, creating a K-length binary word. This word will be used as an index to the voting table V to extract the required output. The wildest classifier will include the encoding of all pixels in the image into a single directory, and then by means of this directory to access a table patching the tags of all conceivable images. Obviously, it is not realistic because of the exponential requires of memory and size of the training set. Hence, instead of describing the entire image using a single transformation, the image is represented as a dense set of patches where individually patch is described by means of the same parameters. Moreover, instead of using a single transformation with a large number of K-binary questions, leading to a huge table, an ensemble of transformations are used, each

with a smaller value of K. Votes of all tables are then summed to give the final output of the transformation. The deep CTE is a model build from several CTE layers, stacked hierarchically to enable the model the ability to learn deep representations.

CTE Drill: CT transformation applies the operation defined below at all positions of the input tensor $T^{in} \in \mathbb{R}^{H_i \times W_i \times D_i}$ in a convolutional manner to form the output tensor $T^{out} \in \mathbb{R}^{H_o \times W_o \times D_o}$. The transformation start with the word calculator function W which applied to a single patch and returning a K-bit codeword. $W : \mathbb{R}^{l \times l \times D_i} \rightarrow \{0,1\}^K$, where l is the patch size. The codeword is computed by K simple bit functions $\{w^k\}_{k=1}^K$, each computing a single bit of the codeword. When fixed, the computed codeword denoted by $W(P;\Theta) = (w^1(P;\Theta^1), ..., (w^K(P;\Theta^K)$. A simple comparison of the difference between two values in the patch and a threshold are used as bit functions:

$$w^k(P;\Theta^k) = T^{in}(p_x + \Delta x_1^k, p_y + \Delta y_1^k, c^k) - T^{in}(p_x + \Delta x_2^k, p_y + \Delta y_2^k, c^k) - th^k \tag{1}$$

The output is a scalar, hence an Heaviside function $Q(x)$ is applied to produce the needed output: $W^k(P;\Theta^k) = Q(w^k(P;\Theta^k) \in \{0,1\}$. Where $\Theta^k = [\Delta x_1^k, \Delta y_1^k, \Delta x_2^k, \Delta y_2^k, th^k]$ are the learned parameters. The codeword computed by the word calculator is used as an index into a voting table $V \in M_{2^K \times D_o}$ to get the D_o-dimensional output $V(W(P;\Theta),:)$ for location P. A CTE layer consists of an ensemble of M ferns $\{W_m, V_m\}_{m=1}^M$. Ferns outputs in each position are summed to get the layer's output:

$$T^{out}[p_x, p_y, :] = \sum_{m=1}^{M} V_m[W_m((p_x, p_y)), :)] \tag{2}$$

The Deep CTE is a stacked model of L CTE layers, connected one over the other. In order to enable gradient based learning, CTE uses a smooth linear sigmoid to replace $Q(x)$, which is not continuous in $x = 0$ and its gradient equals to zero everywhere else:

$$q(x;t) = \begin{cases} 1 & x > t \\ (t+x)/(2t) & -t < x < t \\ 0 & x < -t \end{cases} \tag{3}$$

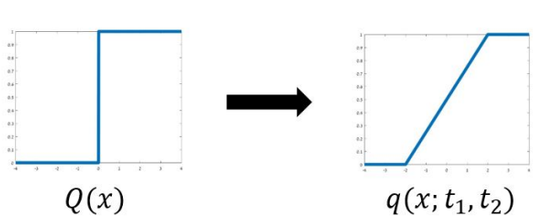

Figure 4: CTE: enabling gradient based learning with a smooth linear sigmoid.

$q(x;t)$ is linear in the section $[-t,t]$ and hence has a gradient in this section. $q(w^k(P;\Theta);t)$ which is similar to a 'soft' bit, with $q(w^k)$ estimating the probability of the bit to be 1 and $q(-w^k)$ the probability to be 0. We now extend it to a soft word calculator $\vec{W}^s : \mathbb{R}^{l \times l \times D_i} \rightarrow R^{2^K}$, assigning each possible word a probability-like value. For each codeword $b \in \{0, ..., 2^K - 1\}$, the activity level is defined by

$$\vec{W}^s(P;\Theta)[b] = \prod_{k=1}^{K} q(s(b,k) \cdot w_k(P;\Theta_k);t) \tag{4}$$

Where $s(b,k) \triangleq (-1)^{(1+u(b,k))}$ is the sign of the k^{th} bit, with $u(b,k)$ denotes the k^{th} bit of the codeword b in the standard binary expansion. $s(b,k)$ is 1 if bit $k=1$, and -1 if bit $k=0$. The probability-like activity level of a word b is defined as the product of the 'probability' of its single bits. The parameter t of the linear sigmoid acts as a threshold. When t is large, most of the bit functions are 'ambiguous'. CTE uses an annealing schedule mechanism, such that t is initiated as $t \gg 0$, set to allow some bit functions values to be in the 'soft zone' $[-t;t]$. t is gradually lowered to achieve a sharp classifier towards the end of the training phase.

4.3 Deep Random Forest

As of today, the two terms deep learning and deep ANNs are used as synonymous. Indeed, deep ANNs has shown great success in various tasks over the last decade and been used in a broad spectrum of application. But is deep ANNs are the only form of deep learning? It is widely recognized that the *'deep'* component, which enables representation learning, is an important part of deep ANNs' success. Deep ANNs are models built based on several hierarchical layers, each containing differentiable non-linear units, that can be trained by backpropagation. Hence, deep ANNs are a single model using the concept of deep learning. Recently, [4] suggested an alternative deep learning model based on a cascade of random forest models.

Random decision forest, or random forest, is an accessible, easy to use and well known machine learning model, thanks to their easy expressibility and various implementations, as well as their empirical success and cheap computational cost. The random forest represent an ensemble of decision trees, each of which is grown separately, using a specific optimization metric for splitting the examples in each of the nodes (such as *gini*, *information gain*, or others [5]). These models are well suited for classification, regression and other machine learning tasks.

Random forest contains a large number of decision trees (can reach to thousands of trees), each performing the same task. The inputs to the random forest is a matrix $X \in \mathbb{R}^{n \times d}$ and the output is a class $Y \in [1,..,C]$ for classification tasks, where n is the number of examples, d is the number of features and C is the number of classes (for other tasks Y will be changed accordingly). The first node in the tree will split X to two subsets according to a given metric. Than, each of the subsets will be split according to different queries. Meaning that there are different query in each node. The input examples will traverse in a different path, reaching to different leafs, which are located at the end of the tree. Each leaf represents a class (in classification tasks) to which the examples that reached it should be classified to. At training time, the class to which each leaf is pointing to, will be decided by the examples that reached that leaf. There are several approaches on how to grow the trees (i.e. how to tune the model's hyperparameters, such as how many splits will each tree apply). For example - for regular random forest, each tree could apply $\sqrt{d}$ splits, where the criteria for splitting is predefined and use different features in each tree (drawn randomly). Another example is the complete random forest, in which a random set of features are drawn, and the splitting at each node is done using a randomized chosen feature. At the end, the output of the trees are grouped together (using a majority vote, for example) and the final output is determined.

The authors in [4] exploit the benefits of random forests to create a deep random forest - a cascade of random forests that enables representation learning as shown in Figure 6. Each level is an ensemble of decision tree forests(ensemble of ensembles). There are different types of forests included to encourage the *diversity*. One of them is completely-random tree forest that contains completely random trees, generated by randomly selecting a feature for split at

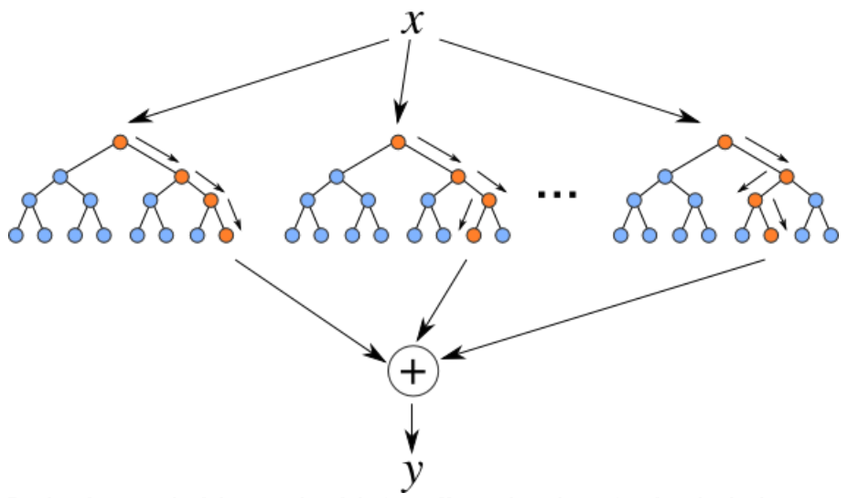

Figure 5: Random forest: each of the examples of the input X goes through a series of randomly chosen queries, until they reach the tree's leafs. To get the final output Y, a grouping metric is applied.

each node of the tree. The other is random forest which contains trees randomly selecting $\sqrt{d}$ number of features as candidate.

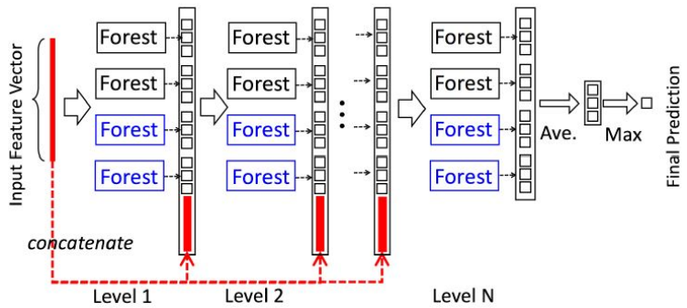

Figure 6: Cascade forest structure example: On each level of the cascade there is two random forests (*black*) and two completely-random tree forests (*blue*). Here, there are three classes to predict. Therefore, each forest will output a 3D class vector, which is then used as an input for the next level.

Each forest will produce an estimate of class distribution, by counting the percentage of different classes of training examples at the leaf node where the concerned instance falls, and then averaging across all trees in the same forest, as illustrated in Figure 7, where red color highlights paths along which the instance traverses to leaf nodes.

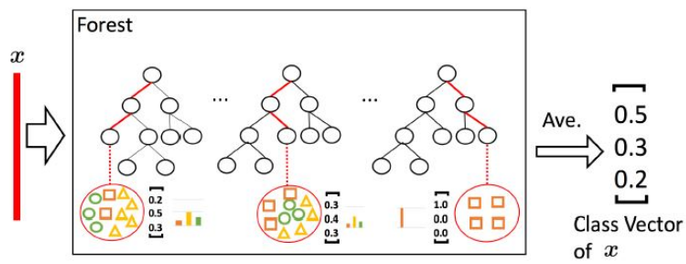

Figure 7: Forest class vector generation example: Different marks in leaf nodes means different classes.

To minimize overfitting risk, class vector produced by each forest is generated by $k-fold$ cross validation. In detail,each instance will be used as training data for $(k-1)$ times, resulting in $(k-1)$ class vectors, which are then averaged to produce the final class vector. After expanding a new level, the performance of the whole cascade will be estimated on validation set.

Multi-grained scanning is applied on cascade forest, meaning sliding windows are used to scan the raw features, see Figure 8. The instances extracted from the same size of windows will be used to train a completely-random tree forest and a random forest, and then the class

vectors are generated and concatenated as transformed features.

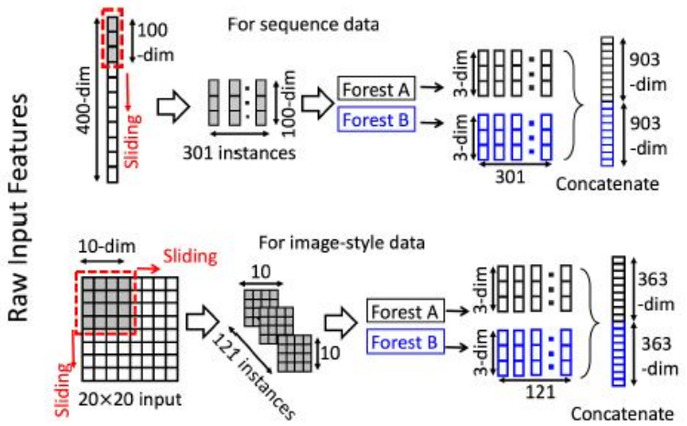

Figure 8: Feature re-representation using sliding window scanning example: 3 calsses, 400-dim raw features and 100-dim sliding window. Note - By using multiple sizes of sliding windows, differently grained feature vectors will be generated.

Final model is actually a cascade of cascade forests, where each level in the cascade consists of multiple grades (of cascade forests), each corresponding to a grain of scanning, as shown in Figure 9. Note that for difficult tasks, users can try more grains if computational resource allows.

Test instance will go through the multi-grained scanning procedure to get its corresponding transformed feature representation, and then go through the cascade till the last level. The final prediction will be obtained by aggregating the four 3-dimensional class vectors at the last level, and taking the class with the maximum aggregated value, as Figure 9 shows.

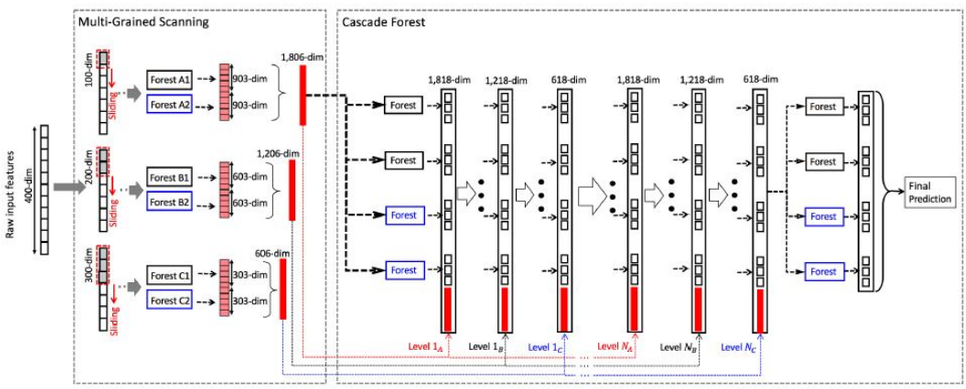

Figure 9: Overall gcForest procedure: 3 classes to predict, 400-dim raw features, and 3 sizes of sliding windows are used.

5 Hierarchical Table Ensemble

The CTE algorithm was design to solve Computer Vision (CV) problems. The input in such problems are images which are spatially arrange. Meaning that near-by pixels are correlated with one-another. Hence the CTE in it's current version, is not well-suited for structured data problems, in which features are not necessarily related. In such cases, the bit functions used in the CTE (see Equation 1) should be changed to get a better features extractor. In this section we will explain how to move from the current CTE algorithm to what we refer as Hierarchical Table Ensemble (HTE).

5.1 From the CT transformation to working with vectors

A the basic neuron of the HTE model is accepting a representation vector $T^{in} \in R^{D_{in}}$ and creating a output representation vector $T^{out} \in R^{D_{out}}$. Similarly to the CT transformation 4.2, the transformation is a pair of a word calculator (W) and a voting table (V). A word calculator W is a feature extractor applied on the input and returning a K-bit index i.e. a function $B : R^{D_{in}} \rightarrow \{0,1\}^K$. The code-word is computed by K simple bit functions $\{w^k\}_{k=1}^K$, each computing a single bit of the code-word.

The computed code-word is denoted by

$$W(x;\Theta) = (w^1(x,\Theta^1), ..., w^K(x,\Theta^K)) \tag{5}$$

Since the input is a tabular data, each feature (cell) is different. Instead of choosing a single cell and compare it to a threshold, the bit functions will be a linear combination of all the features:

$$w^k(x;\Theta^k) = U^k x - th_k \tag{6}$$

Where $U \in R^{D_{in}}$ is a weights vector. This way, the bit function can account for all the features in the input. Notice that the output of such bit function is a scalar, hence an Heaviside function $Q(x)$ (thresholding at 0) is applied to produce the needed output: $W^k(x,\Theta^k) = Q(U^k x - th_k) \in \{0,1\}$. The code-word computed by W is used as an index into a voting table $V \in M_{2^K \times D_{out}}$ to get the D_{out}-dimensional output $V(W(x,\Theta),:)$.

A single layer in the HTE model consists of an ensemble of M ferns $\{W_m, V_m\}_{m=1}^M$. Ferns outputs are summed to get the layers' output:

$$T^{out} = \Sigma_{m=1}^M V_m[W_m(x,\Theta),:] \tag{7}$$

5.2 Bit functions

As stated, the bit functions will be a linear combination of the input, compared to a threshold. At the beginning of the training process, the weights will be set randomly, but as the training progresses, we want to choose only the relevant features for the decision (for each of the bit functions in each fern). Hence we will apply a softmax function with a temperature parameter to control the importance of each of the weights:

$$U = SoftMax_\beta[B] \quad u_j = \frac{exp(\beta \cdot b_j)}{\Sigma_{i=1}^{D_{in}} exp(\beta \cdot b_i)} \tag{8}$$

Notice that $u_j \in (0,1) \ \ \forall i$ and $\Sigma_{j=1}^{D_{in}} u_j = 1$. This way, the weights act as a probability vector over the input. B are the learnable parameters of the model and represents the contribution of each cell in T^{in}.

6 Experimenting With Synthetic Data

In order to successfully debug the HTE algorithm, we conducted a number of experiments with synthetic data - data which was generated and divided to classes with built-in properties. All experiments are simple to compute and their successful results are foretold. Experiment will be consider successful if the algorithm will achive wanted results. If not, it will indicate that

the issue is somewhere in the script and it must be resolved. For all experiments data size is 100000, classes ratio is 0.5 and train test split factor is 0.8.

6.1 Experiments

A In the first experiment we will consider 2-Dim Gaussian random variables from 2 classes. $X_i \sim N(\mu_i, \sigma_i)$, where $\mu_1 = (0,0)$, $\mu_2 = (S,0)$, and $\sigma_1 = \sigma_2 = I_2$. In our experiments we consider a *Separation* factor S which will be the distance between the classes means and will determine the separability of the two classes. In this experiment, separation is possible using a single bit function and a single fern. 'A' is an easy experiment because it is considering 2-Dim variables from 2 classes and S distance considers only one dimension. When $S = 5$ the distribution would be wider(see Figure 11) and we expect the algorithm to reach high accuracy. When $S = 2$ the distribution would be denser(see Figure 12 and we expect to have less accurate results. The optimal solution is the bit function $\Theta(x_1 - \frac{S}{2})$ because x_2 do not contains information about the label.

B Now, instead of 2-Dim data, we experiment with $X \in R^d$. $\sigma_i = I_d$ for $i = 1, 2$ and $\mu_1 = (0, 0, ..., 0)$, $\mu_2 = (K, 0, ..., 0)$. This experiments tests if the informative dimension can be chosen among many d noisy ones. The solution is the bit function $\Theta(x_1 - \frac{S}{2})$.

C Like 'A', but $\mu_2 = [K, K]$ for class 2. This experiment has a medium level of difficulty but a single bit function in a single fern should suffice. Here, the HTE should consider using both dimensions to be able to correctly classify the examples(see Figures 13, 14). This experiment is most significant from the rest because the linear separator considering more than one dimension unlike experiment 'A', and it is important for testing later complex multi-dimensional 'real life' data.

6.2 Initial Results

A HTE algorithm with one bit-function and one Fern reached expected results: Accuracy is 99.36% when $S = 5$ which is greater than 84.42% when $S = 2$. The reason for the accuracy difference is the density of the variables - the more denser it is, the harder is for the vertical separator line $\Theta(x_1 - \frac{S}{2})$ to label the variables correctly. But still the results are very good as expected due to 'alphas' converges and not too high or low learning rate.

B HTE algorithm with one bit-function and one Fern reached expected and the high rate of accuracy indicate that the informative dimension chosen between many d noisy ones. Accuracy for $S = 2$ converge to 83.61% and for $S = 5$ converge to 99.33%. The reason for the difference is the same as the above with just one difference which is the dimensions size of the variables(all noise but one).

C HTE algorithm with one bit-function and one Fern reached expected results and distinguished the two classes, when $S = 2$ the best accuracy was 85.08% and when $S = 1.5$ the best accuracy was 91.88%. The accuracy didn't converge due high learning rate(see Figure 10) but still make high results meaning the linear separator set somewhere close to $x_2 = S - X_1$ as expected.

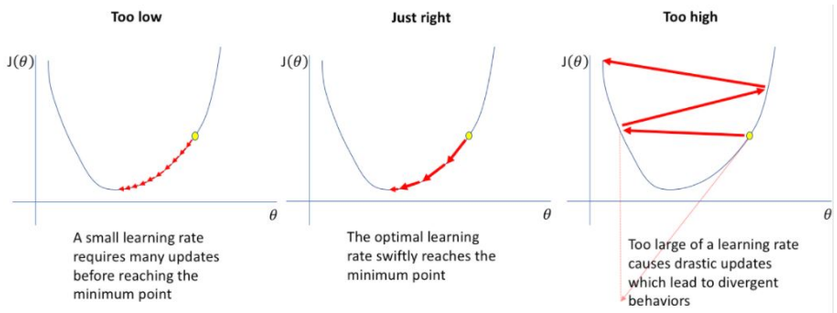

Figure 10: Learning Rate for gradient descent: A hyperparameter that used to train the network. Too low, training will make very tiny updates to the weights which cause slow learning progress. However, if set too high, undesirable divergent behavior may occur in the loss function.

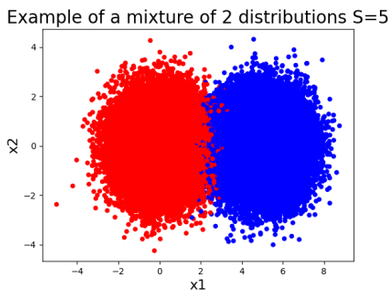

Figure 11: Experiment A 2-Dim distribution, separation factor is 5

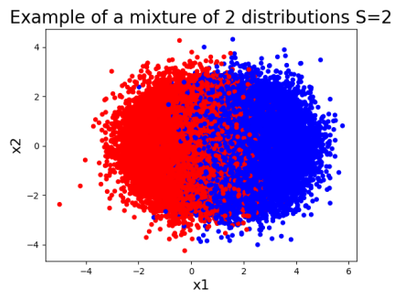

Figure 12: Experiment A 2-Dim distribution, separation factor is 2

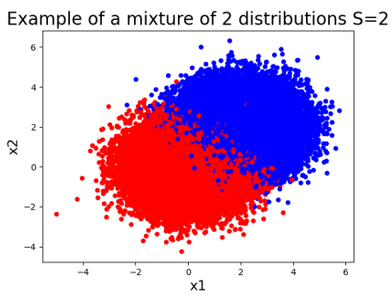

Figure 13: Experiment C 2-Dim distribution, separation factor is 2

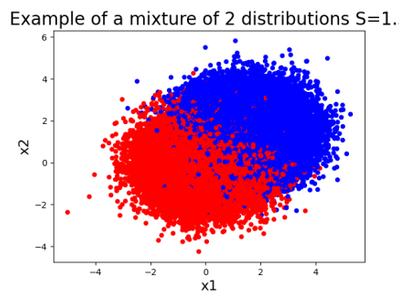

Figure 14: Experiment C 2-Dim distribution, separation factor is 2

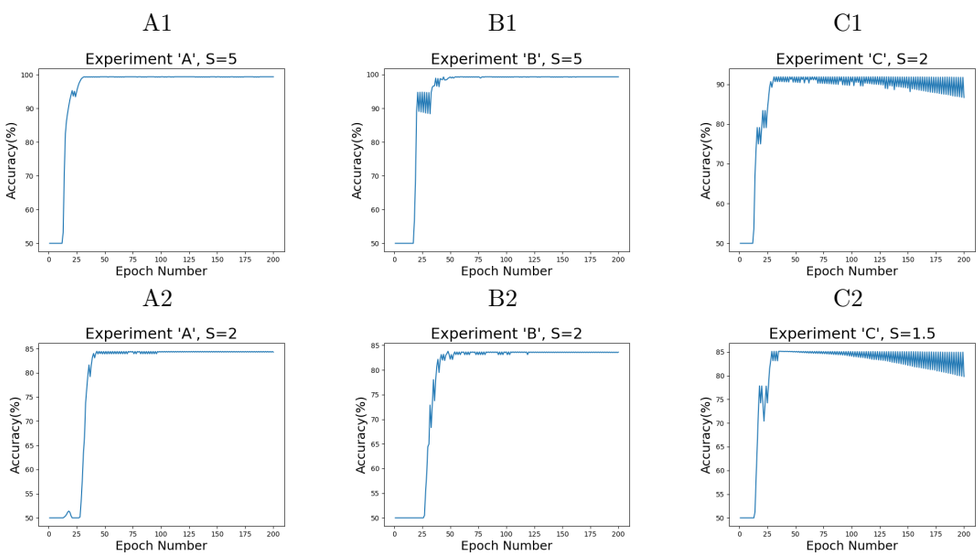

Figure 15: Accuracy by epoch number for all experiments.

7 Experimenting With Real Data

In effort to successfully test HTE accuracy performance we compare HTE results to the best reported algorithm, which is the deep forest, and thus we are using the datasets they used(i.e. 'gcForest', 'Random Forest', 'MLP').

The relative data sets are:

Data set	Number Of Features	Number Of Samples	Classes	URL
LETTER	16	20,000	26	Link
ADULT	14	32,561	2	Link
YEAST	8	1484	10	Link
WINE	13	178	3	Link
IMDB	1	50,000	2	Link

Table 1: Table to test captions and labels

In order to run HTE with these data sets, we preformed data preparation and analysis, as shown below.

7.1 LETTER Data Set

The first tabular data set we examine is 'Letter Recognition' which is a database of character image features. The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

Attribute Information:

Attribute	Information	Attribute	Information
letter	capital letter (A to Z)	x2bar	mean x variance (integer)
x-box	horizontal position of box (integer)	y2bar	mean y variance (integer)
y-box	vertical position of box (integer)	xybar	mean x y correlation (integer)
width	width of box (integer)	x2ybar	mean of x x y (integer)
height	height of box (integer)	xy2bar	mean of x y y (integer)
onpix	total # on pixels (integer)	xedge	mean edge count left to right (integer)
xbar	mean x of on pixels in box (integer)	xedgey	correlation of x-edge with y (integer)
ybar	mean y of on pixels in box (integer)	yedge	mean edge count bottom to top (integer)
		yedgex	correlation of y-edge with x (integer)

Table 2: LETTER data set attributes information

First five samples:

	letter	xbox	ybox	width	height	onpix	xbar	ybar	x2bar	y2bar	xybar	x2ybar	xy2bar	xedge	xedgey	yedge	yedgex
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

Table 3: LETTER data set first five rows

All of the data is non-null numeric values, no more modification is require in order to start the HTE algorithm.

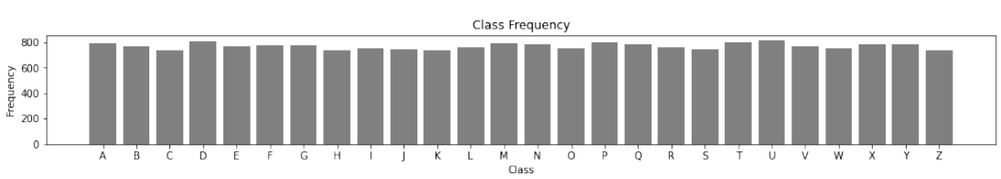

Figure 16: Demonstration of class(letters) distribution

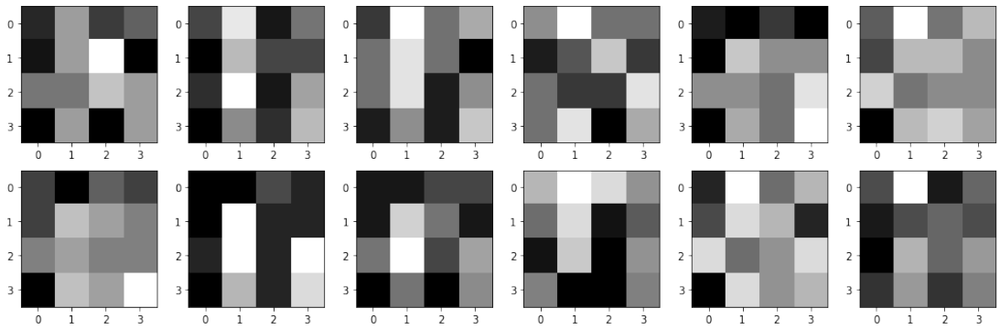

Figure 17: Illustrate the Black-and-white rectangular pixel displays some of the 26 capital letters in the English alphabet

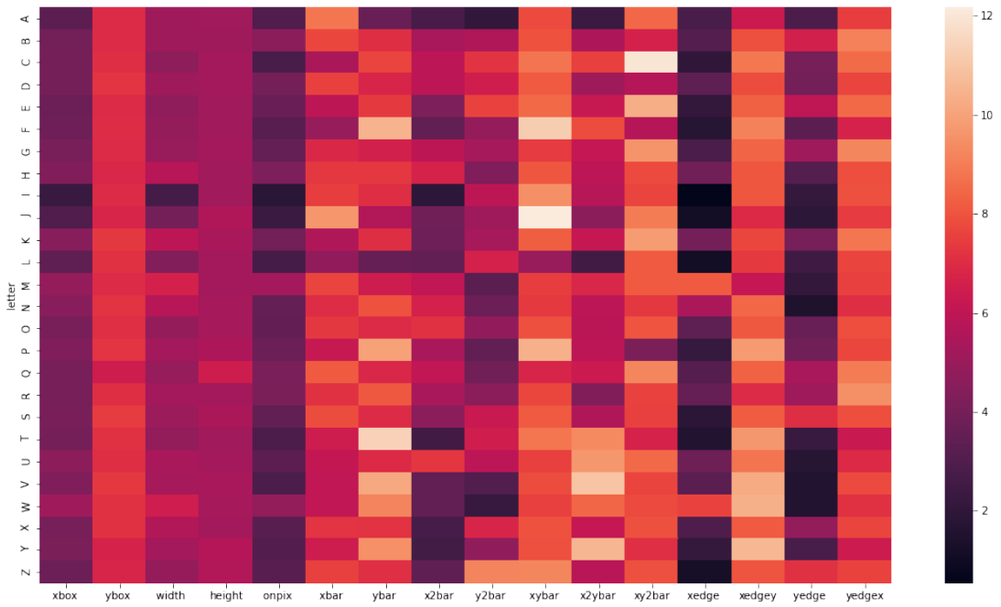

Figure 18: Heat map showing relationship between each letter and each attribute

Its noticeable that 'M' and 'W' have the highest 'xedge' feature values. It make sense since 'M' and 'W' are relatively "wide" at the 'x' axis (pixel-to-pixel) compare to other letters. It can be assumed that 'xedge' feature will effect the letters 'M' and 'W' classification at some degree.

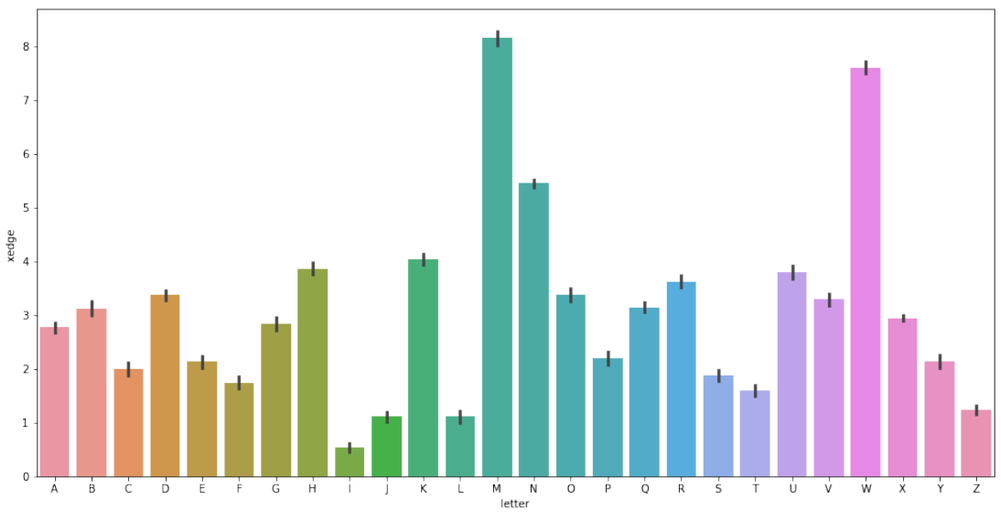

Figure 19: Bar plot showing relationship between each letter and 'xedge' feature

Different approach to see 'xedge' is by the bar plot above. 'xedge' feature values for 'M' and 'W' letters are far from the rest.

HTE Tests LETTER with ResFern & Batch Norm:

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	50	7	50	ADAM	YES	YES	96.10%
3	80	7	50	ADAM	YES	YES	92.975%
4	80	7	50	ADAM	YES	YES	93.825%

Table 4: HTE results LETTER with ResFern & Batch Norm

7.2 ADULT Data Set

Prediction task is to determine whether a person earns more or less than 50K a year based on census data. Also known as "Census Income" data set. Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using certain conditions.

Attribute Information:

Attribute	Information	Attribute	Information
age	person age(continuous)	occupation	labor options(categorical)
workclass	person's work class (categorical)	relationship	relationship status (categorical)
fnlwgt	final weight (continuous)	race	person's race (categorical)
education	type of education acquired(categorical)	sex	gender (categorical)
education-num	number of education years (continuous)	capital-gain	(continuous)
marital-status	(categorical)	capital-loss	(continuous)
hours-per-week	(categorical)	native-country	(categorical)

Table 5: ADULT data set attributes information

First five samples:

age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
90	?	77053	HS-grad	9	Widowed	?	Not-in-family	white	female	0	4356	40	United-States	<=50K
82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	white	female	0	4356	18	United-States	<=50K
66	?	186061	Some-college	10	Widowed	?	Unmarried	black	female	0	4356	40	United-States	<=50K
54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	white	female	0	3900	40	United-States	<=50K
41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	white	female	0	3900	40	United-States	<=50K

Table 6: ADULT data set first five rows

In order to make HTE operational on ADULT data set we had to make some adjustments:

1. Replacing "?" cells with NANs and fitting it accordingly, i.g. "?" in work class was switched to "Never-Worked". Other missing values are imputed with the most frequent value in column or mean if column is numeric.

2. Dropping attributes that does not contribute to the analysis, i.g. 91.671% of all samples are zeros at capital.gain , and 95.335% of all samples are zeros at capital.loss.

3. Handling categorical values with different approaches:

   (a) **One-Hot Encoding** - Using "dummy variables" which are binary attribute(1/0) for each option in each categorical attribute.

   (b) **Label Encoding** - converting the labels into numeric form so as to convert it into the machine-readable form.

Prepared Label Encoding ADULT Data Set Insights:

12 modified numeric attributes for 32,554 samples.

The above histogram shows that age:

- Age attribute is not symmetric.
- it is right-skewed probably because employment rate is higher among younger adults.
- Minimum and Maximum age of the people is 17 and 90 respectively.

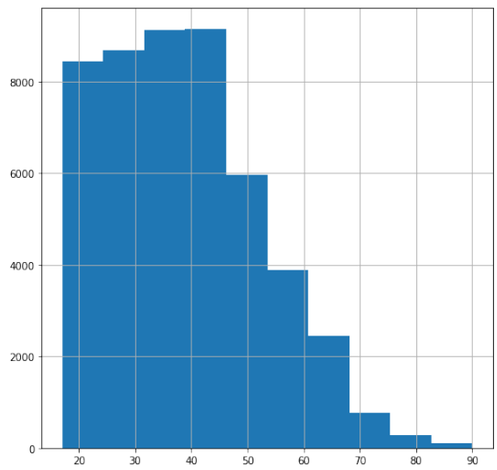

Figure 20: Age distribution

- This data set has fewer observations (540) of people's age after certain age i.e. 70 years.

This can imply that the algorithm should take in consideration that age feature in older samples are less reliable because they represent about 1.5% of total samples.

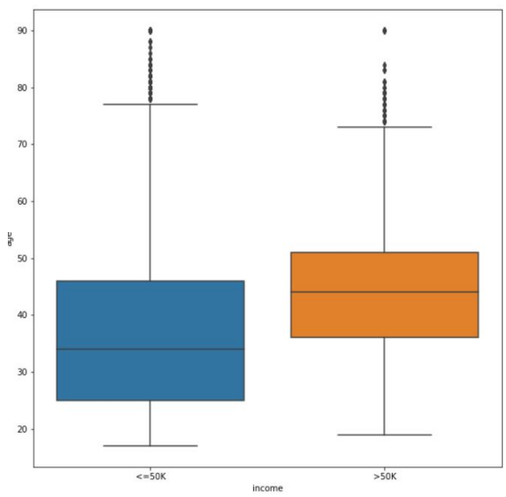

Figure 21: Age-Income relationship box plot

The above bivariate box plot shows :

- The mean "age" for Income group(<=50K) is 36.8 years. And for Income group(>50K) is 44.2 years
- Outliers present in both the income group(<=50k and >50k) wrt "age" attribute.
- Income group(<=50k) has lower median age(34 year) than the Income group(>50k) which has median "age"(42 year).
- Interquartile range(IQR):
  - For Income group(<=50k) , IQR is between [25,46] (long range). Middle 50% of the Age is spread over longer range for the income group who earn ¡=50k.
  - For Income group(>50k) , IQR is between [38,50] (shorter range).
- This data set has fewer observations (540) of people's age after certain age i.e. 70 years.

Hypothesis test the relationship between Income & Age Two sampled T-test :The Independent Samples t Test or 2-sample t-test compares the means of two independent groups

in order to determine whether there is statistical evidence that the associated population means are significantly different. The Independent Samples t Test is a parametric test. This test is also known as: **Independent t Test**.

- Null Hypothesis : there is no difference in Mean age of income group >50k and income group <=50k.
- Alternate Hypothesis :-there is difference in Mean age of income group >50k and income group <=50k.

Results:

$t_{test} = 5.597908016307065$

$p_{value} = 7.635785041686396e - 08$

We reject H_0

We can conclude that there is a significant difference in the mean ages of income group >50K and income group <=50K. It means that age has some contribution to the distinguish income groups.

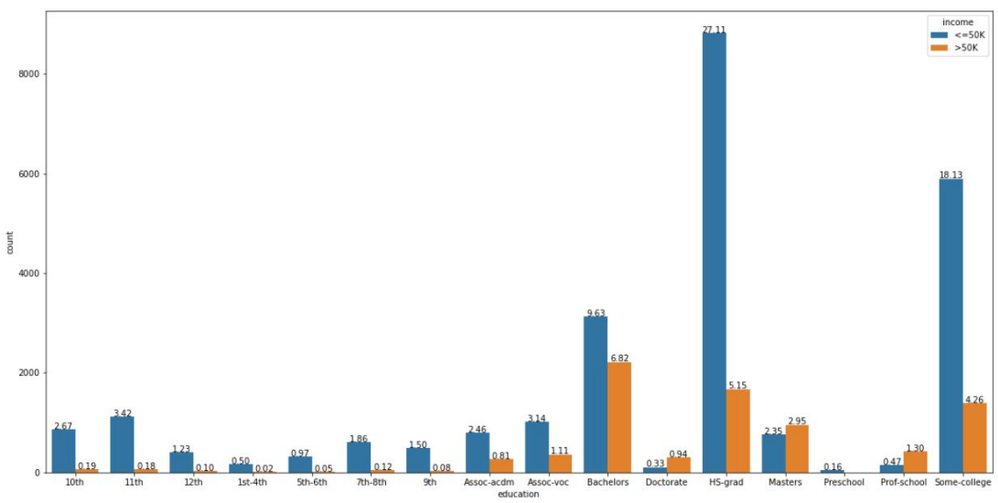

Figure 22: Education-Income relationship

This plot shows:

- The relationship between education attribute and person's income.
- Despite the fact that most of the categories fall under the HS-grad but the interesting thing is only 5.12% of all people belong to the income group 1(i.e. earns more than 50k), surprisingly less than the categories fall under the Bachelors which is 6.78%.
- There only few categories in "education" attribute whose percentage to fall under income group 1 is greater than the falling under income group 0.
- These are prof-school, masters and doctorate.
- We can also infer that higher education may provide better earnings.

Hypothesis test the relationship between Income & Education

- Null Hypothesis: There is no relationship between education and income.
- Alternate Hypothesis: There is a relationship between education and income.

Results:

probability=0.950, critical=18.307, stat=14.104

Independent (fail to reject H_0)

Chi-Squared test concludes:

- As we have accepted the H_0, that there is no relationship between these two categorical variable.
- We can conclude that is **no** dependency of education attribute on the target variable income.

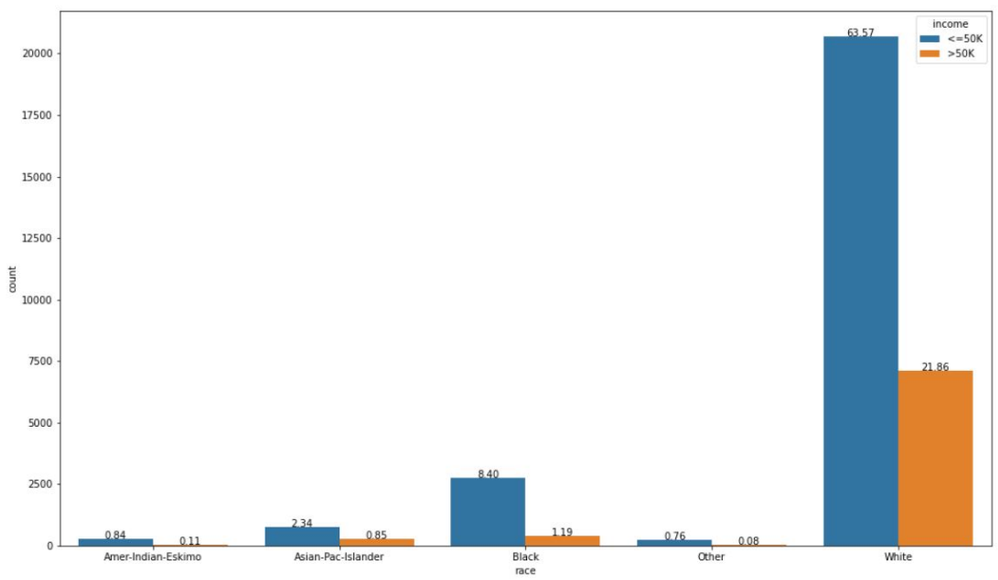

Figure 23: Race-Income relationship plot

This plot shows that:

- The relationship of "white" race with "income" can easily guess based on previous summary statistics.
- There is huge difference between the percentage of fall either groups for each "race" except for the "other"(.63%) and "amer-indian-eskimo"(.74%) but this could be due the lesser number of observations for those categories.

Hypothesis test the relationship between Income & Race

- Null Hypothesis: There is no relationship between race and income.
- Alternate Hypothesis: There is a relationship between race and income.

Results:

probability=0.950, critical=5.991, stat=1.157

Independent (fail to reject H_0)

Chi-Squared test concludes:

- As we have accept the H_0, that there is no relationship between these two categorical variable.
- We can conclude that is **no** dependency of "race" attribute on the target variable income.

HTE Tests ADULT - LABEL ENCODING:

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	50	7	50	ADAM	NO	NO	83.179%
2	50	7	50	ADAM	NO	NO	83.963%

Table 7: HTE Result ADULT Regular

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	50	7	50	ADAM	YES	NO	83.671%
2	50	7	50	ADAM	YES	NO	84.270%

Table 8: HTE Result ADULT ResFern

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	50	7	50	ADAM	YES	YES	83.655%
3	50	7	50	ADAM	YES	YES	84.208%

Table 9: HTE Result ADULT ResFern and Batch Norm

7.3 WINE Data Set

The objective of this data set to cluster different types of wines. This data set contains the results of a chemical analysis of wines grown in a specific area of Italy. Attributes are: Alcohol, Malic acid, Ash, Alcalinity of ash, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines and Proline.

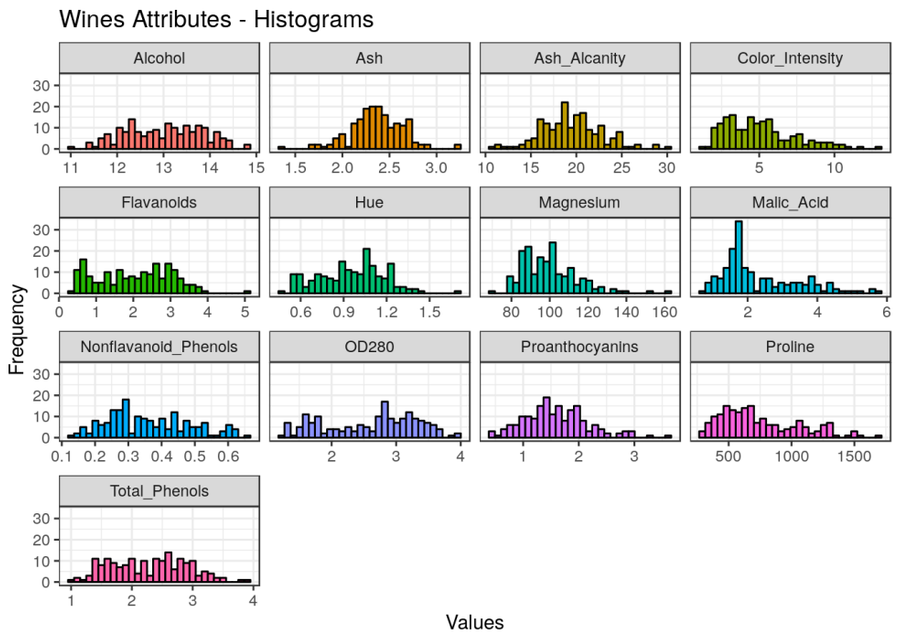

Figure 24: WINE attributes histograms to explore and visualize the data.

The plot at fig.25 suggest that there is a strong linear correlation between "Total Phenols" attribute and "Flavanoids" attribute. We can model the relationship between these two variables by fitting a linear equation, see visual at fig.26 and HTE results below table.12.

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	80	8	15	ADAM	NO	NO	67.084%
2	80	8	15	ADAM	NO	NO	64.263

Table 10: HTE Result WINE

7.4 YEAST Data Set

The classification objective is to predict localization site of protein(non-numeric).
Although we did not preformed data analysis, we did managed to test the algorithm on this data set.

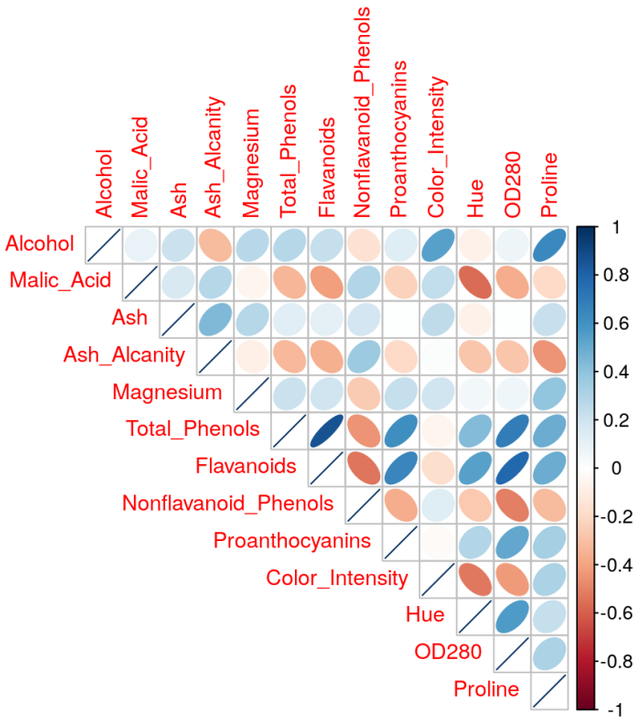

Figure 25: WINE correlation matrix to visualize relationship between the different attributes.

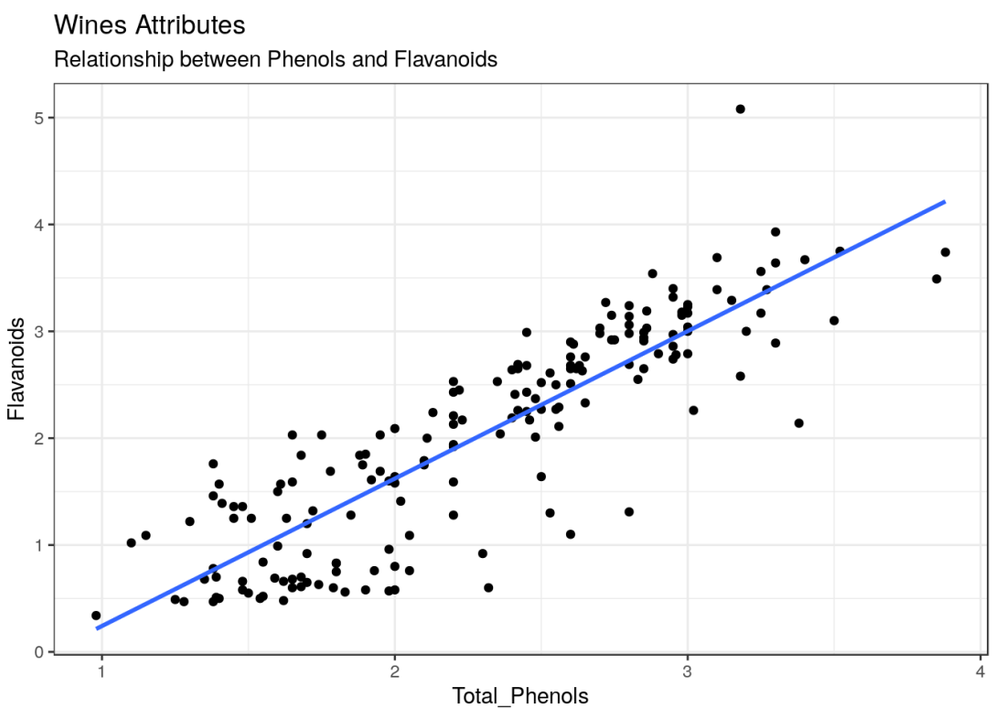

Figure 26: WINE Relationship between Phenols and Flavanoids: We can conclude that is some dependency between the two attributes.

Attribute	Information
Sequence Name	Accession number for the SWISS-PROT database
mcg	McGeoch's method for signal sequence recognition
gvh	von Heijne's method for signal sequence recognition
alm	Score of the ALOM membrane spanning region prediction program
mit	Score of discriminant analysis of the amino acid
erl	Presence of "HDEL" substring(Binary)
pox	Peroxisomal targeting signal in the C-terminus
vac	Amino acid content score of vacuolar and extracellular proteins
nuc	Score of signals of nuclear and non-nuclear proteins

Table 11: ADULT data set attributes information

Layers	Epochs	Bit Functions	Ferns	Optimizer	Res-Fern	Batch Norm	Accuracy
1	80	6	20	ADAM	NO	NO	57.894%
2	80	6	20	ADAM	NO	NO	60.526%

Table 12: HTE Result YEAST data set

8 Final Results

The next table summarize HTE accuracy performances we compared HTE's best results on the data sets above and compared it with other known Deep Learning algorithms 13:

	LETTER	ADULT	YEAST
gcForest	97.40%	86.40%	63.45%
Random Forest	96.50%	85.49%	61.66%
HTE	**96.10%**	**84.27%**	**60.52%**
MLP	95.70%	85.25%	55.60%

Table 13: Algorithms Results On Data sets

We can conclude that the accuracy rate of the HTE is approximately the same comparing to other known Deep Learning algorithms. We believe more can be done (see Future Work) in order to maximize HTE accuracy results and perhaps be a better alternative for known Deep Learning algorithms.

9 Summary and Conclusions

HTE algorithm made to offer alternative approach for Deep Learning on tabular data in matter of processing time and less computing capabilities. We tested synthetic data to show algorithm capability to distinct classes for the classification task, and compared results on known data sets to other known algorithms. Current results shows that HTE is close in accuracy to Forests algorithms and almost better in all tests from MLP. Optimizing the algorithm accuracy preferences is yet to come, but even without it we honestly believe HTE has the potential to enable CPU-based lean devices to achieve Deep capabilities with minimum computing power.

10 Future Work

In the near future we attend to optimize the HTE performances in accuracy and even in matter of processing time.

- Maximizing current data sets results by optimizing the algorithm components, i.g. in the annealing mechanism and all sorts of tools and techniques for the benefit of optimizing the algorithm accuracy rate. Until now, 'ResFern' has been used which is a modification of the known 'ResNet' to the HTE and the methods applied in it. In addition, a normalization batch technique was also implemented, which resulted in a significant improvement in the accuracy rate, so we recommend examining additional methods from the Machine Learning field to improve the learning process in the algorithm in an effort to maximize the accuracy rate. As part of this optimization process we will examine the effect of different hyper parameters values (i.e. number of layers, batch size, learning rate, etc.) while adjusting new optimization tools.

- Implementing the algorithm for CPU-based devices to demonstrate its low computational cost. As we recall, the HTE is aimed to be a low computing deep learning algorithm so it is essential to examine its performances on a CPU-based lean device. For that to happen

the code should be adjusted to this kind of device. While adjusting the code, we will test the accuracy and run-time performances in order to reach maximum accuracy and minimizing resource usage.

- Developing HTE architecture to solve problems with time series data. Time series data can be phrased as supervised learning. Time-Series contains temporal dependencies that cause two otherwise identical points of time to belong to different classes. This generally increases the difficulty of analysing this kind of data. Existing techniques usually depended on adjusted features that were expensive to create. Once the relative dataset is prepared by converting a time series to a supervised learning problem, we must be careful in how it is used to fit and evaluate a model. HTE next mission is to be able to deal with this kind of problem (time series data) and be able to evaluate a model.

References

[1] Aharon Bar-Hillel, Eyal Krupka, and Noam Bloom. Convolutional tables ensemble: classification in microseconds. *arXiv preprint arXiv:1602.04489*, 2016.

[2] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[3] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. Optimizing {CNN} model inference on cpus. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 1025–1040, 2019.

[4] Zhi-Hua Zhou and Ji Feng. Deep forest. *National Science Review*, 6(1):74–86, 2019.

[5] Chris Drummond and Robert C Holte. Exploiting the cost (in) sensitivity of decision tree splitting criteria. In *ICML*, volume 1, 2000.