POLITECNICO DI TORINO

Master degree course in Embedded System

Master Degree Thesis

Exploring Neural-symbolic Integration Architectures for Computer Vision

Fast and efficient Machine Learning Algorithms based on Hyperdimensional Computing for Image and Activity Recognition

Supervisors:
prof. Andrea Calimera
prof. Luca Benini

Graduate Student
Adriele Burco
ID: 235919

University Tutor
ETH Zurich
D.Eng. Abbas Rahimi
D.Eng. Lukas Cavigelli

Academic Year 2017-2018

Summary

The aim of this thesis is to design an efficient and fast algorithm for image recognition to be embedded in low-power devices. By using the brain-inspired hyperdimensional computing (HD), the input image is directly projected into the binary space where all the computations are performed by the cheap xor-popcount operator. The HD alone cannot compete with the state of the art, for this reason some feature extractors have been added to the HD.

The thesis is composed of six chapters. The first one is introductory and describes the state of the art, the related issues and how the HD could be exploited for Computer Vision applications. Chapters 2-4 explore the accuracy-complexity solution space. Each chapter deals with a new architecture while the complexity is gradually increased: it begins with a general description of the background, it continues with the description of the method and finally it ends with a careful analysis of the results by giving reasons behind failing experiments and promising methods. Chapter 2 is a direct application of HD on rough data, which is optimal for simple dataset but it does not scale well to harder ones. Chapter 3 deals with an unsupervised training of image filters, where we focus on a distance preserving binarization technique of the filter outcome and on the spatial correlation between patches. In Chapter 4 an efficient method for the binarization of the last layer of a Binirized Convolutional Neural Network is proposed, which turns to be useful also in the analysis of hidden layers. Chapter 5 explores instead the activity recognition domain and it explains the insurmountable issue related to the frame sequencing by using the HD computing. In the last chapter, an overall view of our experiments and few thoughts on future work are given.

Acknowledgements

I would first like to thank my thesis advisors Abbas Rahimi and Lukas Cavigelli for steering me during this project, for their valuable comments and advice. I would also like to acknowledge the IIS department of ETH for providing me enough computational power to complete my experiments, which otherwise would have taken me an age. I am also thankful to Politecnico di Torino for funds which made this real learning and genuine experience possible.

Finally, I must express my very profound and sincere gratitude to my family and my girlfriend for their unfailing support and unconditional love throughout my years of study. This accomplishment would not have been possible without them.

Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Artificial Intelligence

Artificial Intelligence is a computer science field which concerns all those algorithms, methods and architectures able to provide electronic machines with the capacities usually belonging to human intelligence. The AI space is very wide. To make my studying area clearer, I will briefly describe the main branches:

1. Reasoning and Problem Solving: based on a symbolic representation of the environment, the algorithm is able to define by Logical Deduction a sequence of actions to reach a desirable state.

2. Knowledge Representation: objects, properties and rules will populate a machine alphabet which is stored and dynamically updated to perform all the tasks, merging the items and producing higher level concepts in a hierarchical way. The right representation is still an open question, but Hyperdimensional Cumputing was proved to be very useful in this field [1].

3. Planning: given a possible future state, the machine should define its goal and take decision to reach and pursue it.

4. Learning: ability of automatically improve its performance with experience. It is perhaps the most studied task because of the direct applications in the

industry. Machines can learn to solve a specific problem, without being explicitly programmed for it. It is necessary when human expertise does not exist or it is unreliable, unfeasible (too many data) or when the solution should be adapted to particular and initially unknown cases. Typical problems are the chess game and image recognition.

5. Speaking and Reading: to understand and to use the human language.

6. Movement and Manipulation: strictly related to robotics: after mapping the surrounding space, the machine is able to move objects and itself.

My work deals mainly with the Learning in Computer Vision field, in particular some core problems such as the recognition of images and activities, which have a large variety of practical applications: autonomous car, face recognition, astronomy, and robotics. Image classification is a quite tricky task which requires a very complex learning algorithm in order to properly identify and classify objects and movements.

The aim is to apply the brain-inspired Hyperdimensional Computing to this kind of algorithms projecting the inputs in a HD space optimal for Knowledge Representation, and at the same time reducing the complexity and the power consumption.

1.2 Computer Vision

Nowadays, Convolutional Neural Networks (CNN) are dominating in Computer Vision. Indeed, they are able to extract many meaningful features from an image by locally applying different sets of filters which are automatically learned during the training phase. Their size depends on the kernel parameter. The number of output channels usually increase with the net deepness. In order to apply all those filters, a high computational effort is required. This is usually provided by one or multiple fast, but power-hungry, Graphic Processing Units (GPUs).

The main prospects for the future of computer vision is to embed these algorithms on small devices, e.g. wearable cameras in the IoT domain. This gives us the capability of learning on fly, which is essential for many emerging applications.

Unfortunately the CNNs cannot be embedded as they are, because of the huge amount of floating point multiplication. The computational time, the energy consumption and the memory of a low-power device would be too high, making the HW implementation of CNN infeasible.
The aforementioned issue inserts the CNNs in the high performance-high complexity corner of the solution domain (figure 4.11). Our goal is to fill in the space of the low complexity solutions, finding new Pareto points; moving from the Binarized CNN [2] to much simpler architecture as a single layer network based on K-means clustering [3].

In this space the Hyperdimensional Computing is indeed placed in the lower-left side. It consumes minimum power with basic bit-wise operation, but it has a too simple training algorithm to extract enough information from the image and therefore the performance is not competitive with the state of the art.

In the following chapters I am going to described and analyze four different architectures: from the basic HD, moving step by step to much more complex ones. For each of them, I will firstly introduce the related background concepts, secondly I will describe in details the architecture, and finally I will analyze the results, focusing on advantages and drawbacks of the HD and the principal factors which impact on the performances.
The main differences between one model and the others are in the back-end, where different algorithms for preprocessing and feature extraction are used, but maintaining the complexity always as low as possible.
The front-end is instead common for everyone: it is always binary in the HD space. In this way, under such conditions, all the HD properties can be applied and the outcomes is eventually used in the field of knowledge representation. Indeed, it would be possible to ask the analogical question "What is the Auto-mobile of Air?" as [4] clearly describes with many other interesting examples.

Chapter 2

HD computing

The first suggested architecture is a straightforward application of HD properties applied directly on input data. Before moving to the details, I shall introduce the Hyperdimensional Computing from a theoretical point of view.

2.1 Background

The huge quantity of neurons and synapses suggests us that our brain needs large circuit for a correct behaviour. Starting from this idea, why should we represent our data in just 32-bit words? Why not using 10 thousand bits? As Kanerva presented in [1] the brain-inspired High-dimensional Computing reserves many good properties such as randomness, positional independence and robustness. Based on the HD computing we can find in literature many application in Language Recognition [5,6], Hand Gesture Recognition [7], classification of EEG error-related [8].

2.1.1 Randomness

The basic word in the HD hyperspace is called hypervector (HV) and it is composed by many bits in the order of thousands. An HV is usually pseudo randomly generated, where "pseudo" refers to the additional constraint of equal density. This

means that a typical HV is composed by an equal number of zeros and ones randomly permuted. With this assumption, all the generated hypervectors are semi-orthogonal between each other. Even if the exact number of orthogonal vectors (with density 50%) is just $\log_2(D) + 1$, there is a huge number of vector which is quite orthogonal. For instance, generating randomly ten thousand of vector with dimensionality 10000, the standard deviation is around 50 and all the generated HV has an amount of equal bits in the range 4800-5200.

2.1.2 Positional Independence and Robustness

Due to the randomness, when an item is associated to an HV, all the information brought by the item is uniformly distributed among all the bits of thia HV. Indeed, even if some bits of the HV are flipped by noise, process variation or else, all the information is still available. For instance, if two HVs 2500-bits apart represent two items, we can flip up to $\frac{2500}{2}$ of the bits of one items, and still we will recognize it by comparison with the original ones even in the worst case. However from a probabilistic point of view we can flip many more bits without computing any mistake with an high probability. Let's define s = percentage of equal bits, $d = 1-s$ is the distance, e = error = percentage of flipped bits, |A*B| = similarity between A and B, A_n = noise version of A. The correct guess is got when $|A_n * B| < |A_n * A|$. Clearly $|A_n * A| = 1 - e$. Probabilistically B has s*e flipped bits that were initially equal to A and (1-s)*e flipped bits that were different, therefore $|A_n * B| = s - s * e + (1 - s) * e$. Then $s - s * e + (1 - s) * e < 1 - e - m$, where m is a margin factor that increase the confidence of the result, because the distance between A_n and B is just the expected value, a proper choice could be the variance. This results in $e < (d-m)/2d$. To give an example, D = 10000, m=0.1, d=0.25, e < 0.3: we can flip up to 3000 bits (e*D) of A and the expected distance from B would be 6000 bits that is at least 1000 bits (m*D) apart from $|A_n * A| = 1 - e = 7000$. This demonstrates that the HD computing is really optimal for encoding noise signals because of its robustness and error tolerant characteristic.

2.1.3 Operations

There are three main operations in the HD domain: bundling, binding and sequencing. They all require cheap component wise computations.

Bundling

Bundling refers to the creation of sets by accumulating different HVs in a single one. The final HV is binarized by a threshold equal to half of the number of items in the set (in the bipolar case it is simply zero). The set generated in this way maintains the information of each item. In fact, if the HV A is bundled to the HV B, then a new HV C, similar to both A and B, is created. Therefore, if an item is compared to a set and a relative high similarity is detected, it means that actually it probably belongs to this set. If the set is composed by random HV it is possible to detect a similarity higher than random case only if the length of the set is limited, otherwise we have a saturation issues. From image 2.1 it is clear that no more than 100 orthogonal HV can be stored in a set of dimensionality 10000 bits. Increasing the dimensionality of HVs, the saturation is reduced and some other HV can be stored in the set.
These considerations are valid only in the case of orthogonal HVs. In the real case, the components of the set are similar to each other; for instance, a class HV which represent the "Car" is composed by many photos of cars. In this case, the saturation is trivial. Indeed, in the extreme case, where all the images are identical, then the set could contain an infinite number of components. In a typical set composed by non-orthogonal items, the saturation manifests itself as "mean" operator over all the components. If a feature is more recurrent than others, then the final set would be much similar to this feature. So the particular cases, which manifest themselves only once or few times (e.g. a vintage car), are usually lost in such a kind of sets. It is impossible to define precisely upper thresholds for the length of sets, because it depends on many factors: amount of recurrent and particular features, main distance from other sets, similarity between the item and the set.

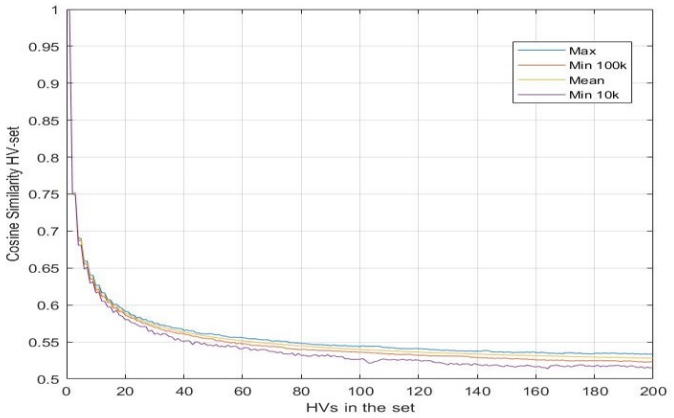

Figure 2.1. As soon as HVs are added to the set, the cosine similarity between the set and all its orthogonal components is reduced. The minimum value is the most meaningful. If it is high enough all the components belonging to the set are correctly identified. Otherwise, if the cosine similarity is below 0.53, then it would be probably identified as a random HV not belonging to the set. Increasing the dimensionality from D = 10k by a factor of ten, the mean value is slightly increased, so that more HV can be added to the set.

Binding

In many cases, it is useful to bind two items together, e.g. if it exists the concept of a car (properly represented by an HV) and the concept of the colour red, we may want to create the object "red car". This is easily reached by binding the two items by a xor component-wise operation (or multiplication in the bipolar case). This kind of operation maintains the original distances of the items, which means: if "red" and "blue" are orthogonal HV, then also "red car" and "blue car" are orthogonal. To make it clear, the binding operation of many objects (colors) to the same HV (car) moves all the target points to a different zone of the HD space as a unique block, without affecting the distances inside the block. A complex vector is produced by binding many basic vectors together, such as a sentence is composed by many words. To give a concrete example, in our architecture we will bind: position, color and value in a single pixel HV.
It is also possible to unbind an HV; one of the beautiful properties of the HD is that the inverse of the xor is always the xor, so $(a * b) * a = b$. Indeed, if any addiction

are performed, we can unbind (a*b) with "a" to get "b" without any added noise. Usually, the binded items are added together, so that the recall of an object will produce noise: $((a * b) + (c * d)) * a = b + c * d * b = b + noise$.

Permutation

"Permutation is the shuffling of the vector components and it can be represented mathematically by multiplication with a special kind of matrix, called the permutation matrix, that is filled with 0s except for exactly one 1 in every row and every column. Because permutation merely reorders the coordinates, the distances between points are maintained just as they are in multiplication with a vector [1]: $\rho X * \rho Y = \rho(X * Y)$ and $d(\rho X, \rho Y) = |\rho X * \rho Y| = |\rho(X * Y)| = |X * Y| = d(X, Y)$". A straightforward and cheap HW implementation of a permutation is just the shifting operator. Sometimes, a random permutation (so no shifting) is used in order to force the bits of an HV to be uncorrelated. This is particularly useful in Holographic Reduced Representation (HRR), which is another Vector Symbolic Architectures (VSAs) which uses convolution instead of component-wise multiplication. In [9] they need to make pixel of the images uncorrelated, because of "convolution memories need the elements of the diagonals to have values that are evenly and independently distributed around zero in order for the irrelevant terms to cancel". For this reason, shuffling is used. Permutations are also used in the aforementioned paper and others [10] for hiding information in holographic representation; this means surrounding a complex HV (a+b) before adding it into a set. In this way the target HV remains a unique entity of the set, and its components are not merged to other ones: $\rho(a+b)+c+d$ instead of $(a+b+c+d)$. They are also used when the order of items matter, e.g. in text recognition and temporal encoding. It will be used in our experiments to encode frame sequences of a video.
Another typical usage of permutation [5, 11] is to encode N-grams in this way: $\rho(\rho A * B) * C = \rho\rho A * \rho B * C$. This efficiently distinguishes the sequence A-B-C from A-C-B, since a rotated hypervector is uncorrelated with all the other hypervectors.

2.1.4 Item and Associative Memories

The HD computing is a symbolic language based on items which are combined to create more complex concepts. Usually these items are semi-randomly generated and initially stored in an Item Memory (iM) containing the basic blocks, sometimes called the alphabet of the architecture, which will produce different words depending on the performed combinations. Instead, the complex HV often represents a higher level concept that is stored in the Associative Memory (AM). The AM therefore will not contain orthogonal HV and hardly ever is used for creating new higher level concept. The main application of an AM is to be compared with the query HV (a never seen HV) and to find the class in the AM which better represents the query.
Many works have explored the Hyperdimensional AM. One of the cheapest HW implementation was based on resistive memories [12]. They take advantage from the robustness of HD, maximizing the inference speed and energy saving at the cost of reading accuracy which minimally affect the classification accuracy if the classes in AM are quite distant from each other.
The Item Memory instead could be generated inline by means of one or multiple LFSR, so that no additional memory is required. This would save much memory, but it probably slightly slow down the inference. Indeed, fixing the seed of the LFSR and moving forward as soon as the address is reached, is slower than a typical memory addressing, especially if the memory has a huge number of cell.
In literature, many works (such as [11]) suggest the use of sparse encoding for energy saving, reducing the density of the HV the number of 1s flipped is reduced as well as the dynamic energy consumption. We did not focus on sparse encoding in our work, but it could be explored in the future.

2.2 Straightforward application on digits

The first architecture analysed in my thesis was the most basic HD technique: one-shot training and value-position pixel binding, using an iM (Item Memory) and a CiM (Continuous iM). A straightforward HD implementation which is really suitable for simple tasks such as digit recognition.

2.2.1 MNIST dataset

The Modified National Institute of Standards and Technology database (MNIST [13]) is a popular dataset consisting of handwritten digits. It is composed by a training set of 60,000 gray-scale images and 10,000 test images. The digits have been size-normalized and centered in a fixed-size 28x28 image; some examples are shown in fig. 2.2. The classification goal is to categorize the handwritten digits from 0 to 9 into 10 different classes.

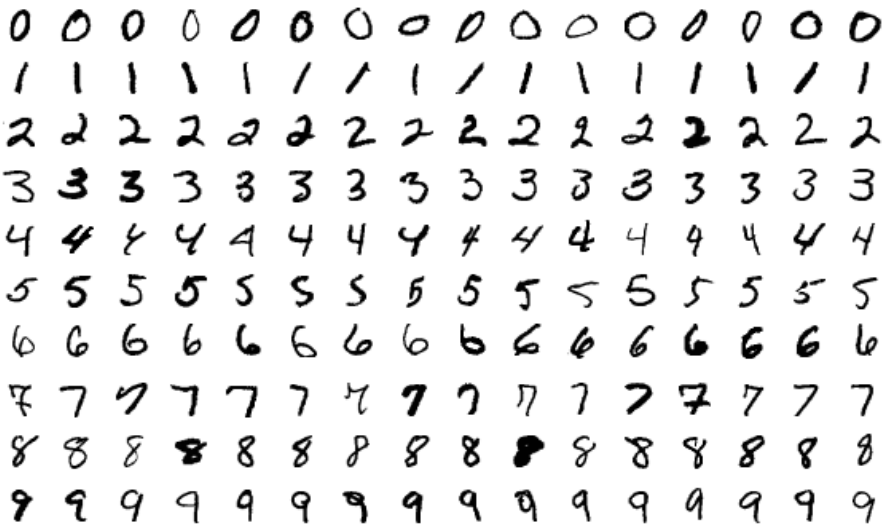

Figure 2.2. MNIST digits examples

2.2.2 Method

In this section, I am going to describe in details this first architecture. An overall view of the pipeline of the learning step is given in fig. 2.3. Both the encoding and classification parts, are completely based on operation with Hypervectors.

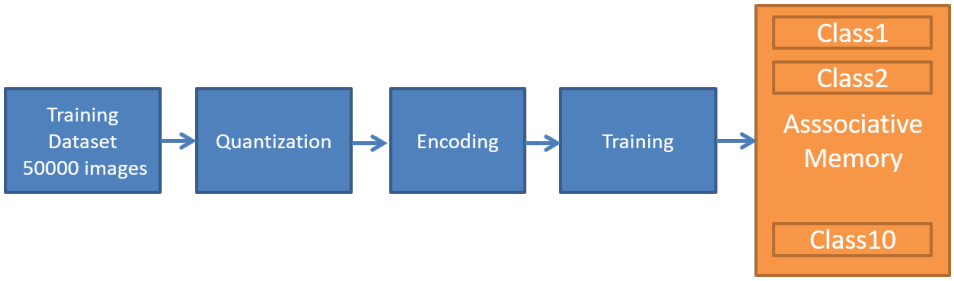

Figure 2.3. MNIST pipeline.

Encoder

Let's start with the encoder shown in 2.4. First of all, the input digits are quantized in m intervals, so the pixel value range is reduced from 0-255 to 0-m. This value is used for indexing a Continuous Item Memory (CiM): in this way the HV corresponding to pixel value 0 (PV1) is orthogonal to PVm, instead all the others PVi are placed at intermediate distances; therefore, closer values match up with lower hamming distances and this will increase as you get further away. For this architecture a dimensionality (D) of 10000 is used for each binary HV, so the orthogonality condition is reached when the hamming distance between two hypervectors is 5000 bits.

The pixel value HV is then binded with its position encoded in another iM. As first experiment, an orthogonal HV (called PP) was associated to each pixel position. In this way we consider uncorrelated positions; later different alternatives will be taken into account: Cim, Sandwitch iM, Positional iM.

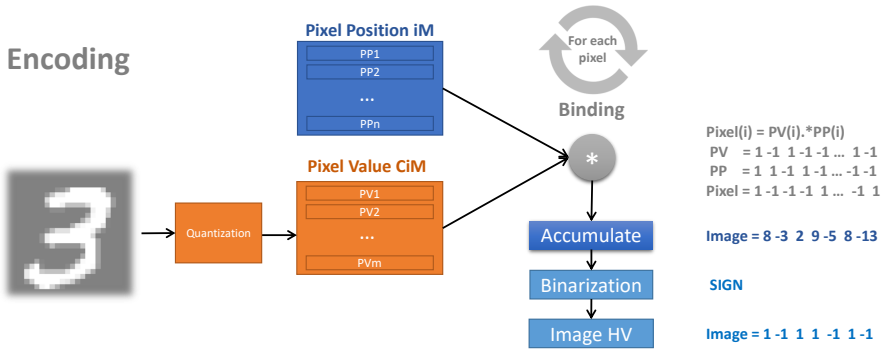

Figure 2.4. The back-end is composed by an encoder which generates a binary Image HV as a linear combination of semi-random HVs. Each HV is stored in two fixed iMs addressed by the position and the quantized value of the pixel.

As already explained, the binding between two bipolar HV is obtained by a bit-wise multiplication between the two vectors. The results would be again bipolar and dissimilar from each factor. The resulting HV represents our pixel. Performing the binding for each pixel and accumulating all the results, a single integer HV is obtained. Finally, the sign of each integer value is used for generating the bipolar digit HV. This final binarization is really important because it protects the internal structure of pixel value-position as discussed in sec. 2.1.3, where permutation is used instead of binarization. Without this step, adding two different image HV in the same class HV, would make it ambiguous which pixel belongs with which image. It would become an agglomerate of pixel value-position couples without any relation with the original images.

Training

When the image HV is obtained, it is sent to the training block which performs a very fast one-shot training (fig. 2.5). This implies an accumulation of all the digit HVs belonging to the same class; after a second binarization step the resulting Class HV is stored in the correspondent cell of the Associative Memory (AM). With this architecture, each class is trained separately so no correlation among classes is considered. The final AM is composed by 10 HVs (one for each digit)

and its memory requirements is equal to 10*D bits = 12.5 kBytes.

Figure 2.5. One-shot training of a Class HV.

In order to further improve the accuracy at the cost of higher computational time and power consumption, we can test multiple times the training set, taking into account all the mispredicted samples. The algorithm is very simple: if the guess for digit X is equal to Y instead of X, then both the Class HV X,Y should be updated. The class Y that caused the misprediction is moved away from the query and the correct one is moved closer to:

1. ClassX = ClassX + query

2. ClassY = ClassY − query

We decided to call this training process HD perceptron (fig. 2.6), which includes a kind of back propagation. It is repeated N epochs, until the training set start over-fitting. For further details on this algorithm refer to the section 2.4.

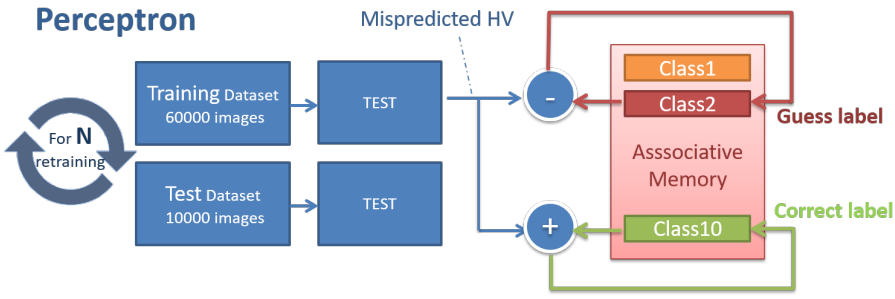

Figure 2.6. HD Perceptron training. For each mispredicted training sample an addiction and a subtraction are performed. In the example, the input label is 10, but the guess was 2, so both the Class HV are updated.

Inference

Once the AM is trained, the inference is straightforward. As shown in fig. 2.7, the query HV is compared against every class stored in the AM; the most similar would represent the guessed class. The distance metric in the binary space is the Hamming distance which is computed by counting the number of equal bits between the two HV, therefore an array of XOR gates and a popcount operator are sufficient.

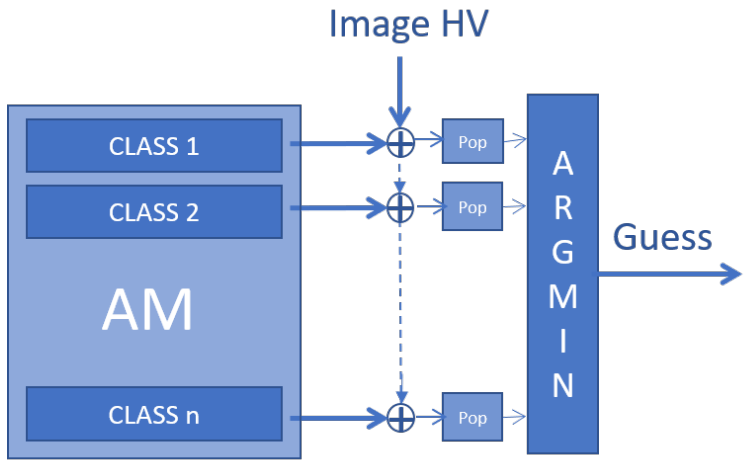

Figure 2.7. The Guess is represented by the class with minimal Hamming distance from the image HV under test.

2.2.3 Analysis

Capacity issue and Recurrence of similar shapes

One of the main issue of this architecture is that each HV has a maximum capacity depending on its dimensionality (D). If too many samples are added to the same HV, it saturates and a kind of mean is performed on the stored information. In this way, the most recurrent digit's shapes will have a predominant weight and the images with particular features are totally lost. In this case increasing the number of bits for each HV would fix the problem at the cost of higher memory

requirements. Nevertheless, if the recurrence of a particular shape is very high, the D parameter would not affect the performances. In order to clarify this concept, we shall consider an extreme case where 51% of all the Image HVs are identical; in this case the information brought by the other 49% of HVs is deleted after the binarization. In fact, the one-shot HD training performs just a bit-wise majority among all the inputs and under such conditions the class HV will collapse into the most recurrent image HV.

A possible solution is to store more HV per class. Let's call them prototypes of a class. The main issue is to select a proper subset of input digits to train each prototype. If the selection is carried on randomly, the problem will probably still be present. Instead, we should group together, as much as possible, all the almost-identical shapes. In this case, some prototypes will be trained with less samples than others, but without any predominant feature. In this way, if the capacity of each prototype is sufficiently high, we can store all information without any particular loss.

As an alternative, the HD perceptron can be used. Indeed the most recurrent digits would be correctly predicted with high probability, so during the epochs only the particular digits are added multiple times to the classes. Therefore, if the number of epochs is high enough, the gap between more and less expressed shapes is reduced. The HD perceptron should converge to a solution where all the shapes are equally present in the Class HV. The training is usually interrupted when the accuracy begins slightly oscillating around a constant value. This means that some of the most recurrent digits start to be mispredicted and so they enter in a kind of loop where the perceptron update let them to be correctly predicted again, but in the next epoch they come back to be mispredicted because of the sequent update.

Quantization levels

Our experiments show that many quantization levels are useless for this task. Moreover, we obtained very good performances in the extreme case of m=2. Consequently, the only two possible colors are black and white. This was possible because of the high contrast level in the digits.

In this specific case, the white pixel corresponds to the background and brings no further information with respect to the position of the black pixel. So the binding of a pixel with 0 value does not increase the accuracy, in fact it adds only noise to the final image HV.

Spatial Correlation

In the first architecture, the Position Item Memory was generated with all the HV orthogonal between each other. Hence, it is assumed that the spatial correlation does not affect the result. This is of course not true. Nevertheless, with a simple dataset, for instance, the MNIST, is not necessary. Scaling this architecture to other datasets, it will be necessary to include some correlation.

Multiple Cluster issue

Another main issue comes from the clustering distribution. The one-shot HD training can be seen as the binary version of the Nearest Mean classifier [14]. Luckily, this is an old known algorithm and all its problems have been already studied. Let's see the main ones:

1. Multiple Prototypes: if the samples of a class are divided into two main cluster, the mean of all the samples would be far away from both. An example is shown in fig. 2.8. If the data has this kind of distribution, the only solution is to associate a prototype for each cluster. It results in a multi-prototype AM, how this cluster are obtained and further information are given in section 2.6.

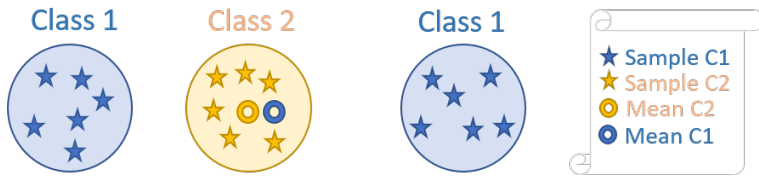

Figure 2.8. The mean of all the samples of class 1 falls into the space of class 2 causing many mispredictions. Moreover, all the samples on the left side would be associated to the nearest mean, so to the wrong class.

2. Attribute Weighting: the higher density of a cluster w.r.t. the others could influence the accuracy, e.g. fig. 2.9. For this reason, each cluster is weighted depending on its relevance, computed as the variance of the sample distances from the centroid. In the HD space it corresponds to adding a bias value for each class in the Associative Memory, nevertheless we did not exploit it because we want to maintain the architecture fully binary.

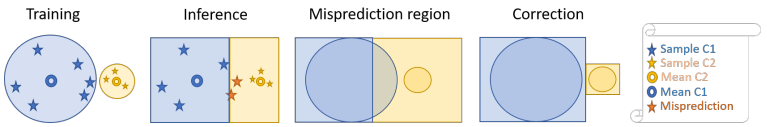

Figure 2.9. The overlapping region between the orange square and the blue circle cause mispredictions. Dividing each cluster by its variance, the small cluster would not influence the biggest one anymore.

3. High Variance along some axis: a cluster could spread into the space in a not uniform way. The distance of the query is therefore weighted by the variance of the cluster in the query direction. In the HD space this would be very expensive because the variance of each direction is needed. To use the total variance only is often sufficient.

2.2.4 Results

On the base of all the previous consideration, we came up with the following setup for the MNIST dataset:

1. Minimum number of color levels: the digits are quantized in two values, just black and white (table 2.1).

2. A strong positional correlation is not necessary: instead of the Position Binding Matrix we used the Sandwich item Memory.

3. Pixel level 0 is not used, it does not add any further information: The correspondent HV in the PV item memory was set to 0 (no effect during training).

4. Increasing the number of prototypes up to 100 gives good improvements if the HD perceptron is not used (table 2.2). Nevertheless the best results are

obtained using the HD perceptron algorithm with one prototype per class (AM composed by only 10 HVs).

m levels	2	3	4	8	16	32	64	100
Accuracy	82.6	82	82.4	81.67	81.63	81.43	81.64	81.34

Table 2.1. The table shows the best number of quantization levels is 2 with the following setup: 32 prototypes per class, Random Position iM. Increasing m up to 100 the Accuracy on cross validation set never overcomes the 82.4%.

# Prototypes	3	4	8	16	32	64	100
Accuracy	80	77.8	80.3	81.9	82.6	85.5	89.1

Table 2.2. The table shows that the number of multivectors in the AM is directly proportional to the accuracy, at least up to 100. We have not increased it anymore because the AM would become too large to be stored. The tests used $m = 2$ and the Sandwich iM.

Applying only the one-shot learning and the aforementioned setting, we reach an accuracy equal to 89.1%, this means a 10% loss with respect to the state of the art. The HW and power cost are minimal. Both the Item Memories can be generated online using dedicated HW (for instance an LFSR) for the random generation of each HV. The trained Associative Memory instead needs to be stored and it requires (# of prototypes)*(D bits) = 100*10000 = 125 KBytes. If we use just one prototype per class, the accuracy drops to 83.7% and the memory is reduced by a factor of 10.

HD Perceptron Results

Firstly, we applied the HD Perceptron without binarizing the scalar AM, and using a constant learning rate equal to 1. This let's the test set leading to 94% of accuracy (fig. 2.10), but the training set is clearly over-fitted due to the great amount of parameters and degrees of freedom. Moreover, using the scalar AM during inference the popcount is no more sufficient and it should be replaced by an adder. Luckily, we do not have to perform any multiplication because the inputs remains bipolar and so we can just xoring the sign bit of each scalar value with the correspondent input bipolar feature.

Replacing the Scalar AM with the Binary AM, we have to take into account the learning rate (LR). In the binary space, the training curve is no more monotone and it is prone to oscillations. A wrong LR would cause divergence or unwanted behavior (fig. 2.11).

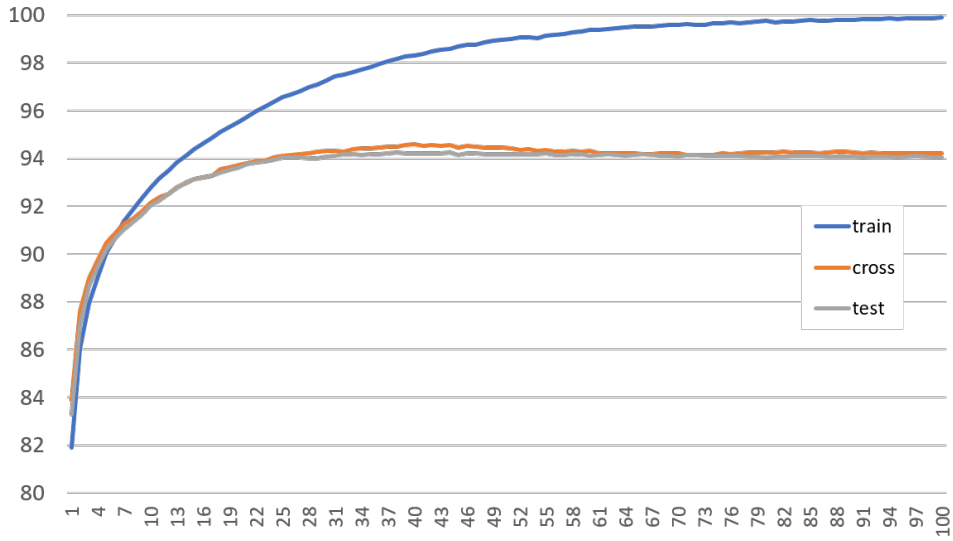

Figure 2.10. Learning Curves for HD perceptron training on MNIST dataset. In this case the weights are not binarized. 100 epochs are shown, it is clear that after the 30^{th} epoch the algorithm starts overfitting. LR=1

The most stable LR is obtained reducing it by a factor of 10 (starting from $LR = 1$) at predefined epochs: [4,30, 100]. If we use $LR = \frac{1}{\sqrt{epoch}}$, the learning curve starts oscillating in the later epochs. The results are plotted in fig. 2.12. It is noteworthy that the Positional Binding is worse than the Sandwich item Memory (further details in section 2.5.2); so, for what concerns the digits, the correlation between pixel position is meaningful only locally (slight permutation).

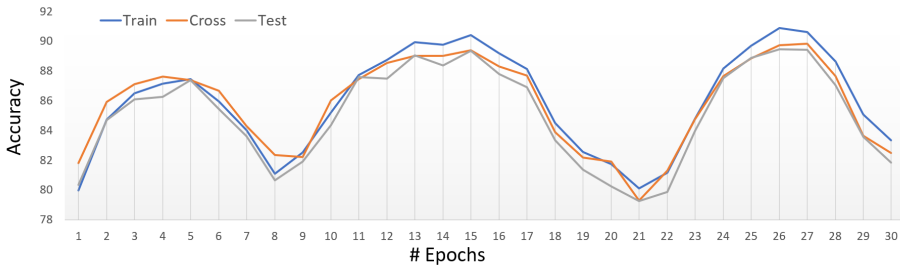

Figure 2.11. The reason of this unstable behavior is the learning rate. It should be reduced after the fourth epoch or from the beginning.

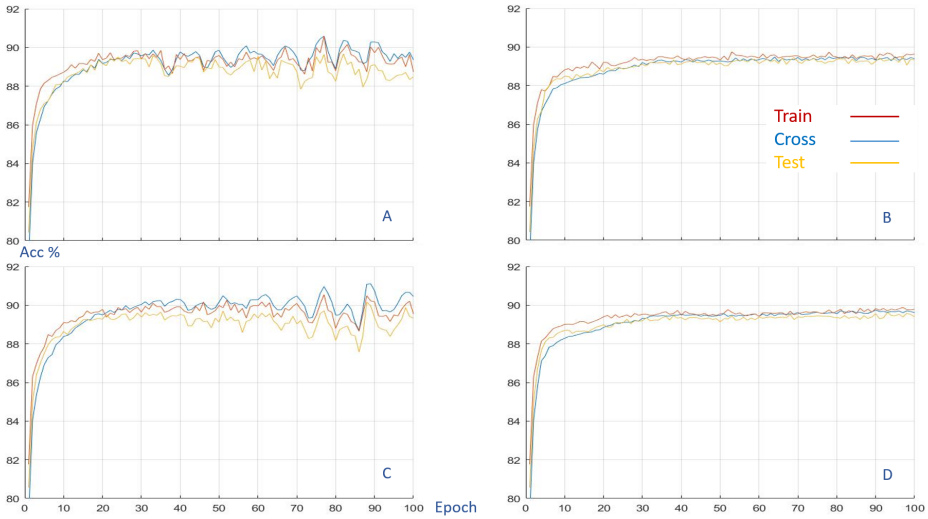

Figure 2.12. Sandwich iM (C,D) gives slightly better results with respect to Positional Binding Matrix (A,B). $LR = \frac{1}{\sqrt{epoch}}$ (A,C) is less stable than $LR = [1, 0.1, 0.01]$ for intervals [0-4, 5-30, 31-100] (B,D).

2.3 Scaling to images

Given the final architecture for the MNIST dataset, we decided to scale to a more complex image dataset such as the CIFAR-10. The front-end remains the same as before, we had just to manage the three channels applying various preprocessing techniques and some modification to the encoding.

2.3.1 CIFAR-10 dataset

The CIFAR-10 dataset [15] consists of 60000 32x32 RGB images equally subdivided into ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck (fig. 2.13). The test set is composed by 10000 images. The images belonging to the same class are often dissimilar from each other, leading into a really complex classification task. For instance, into the ship class it is possible to find a canoe full of people with their hands up which is totally different from the usual boats (fig. 2.15).

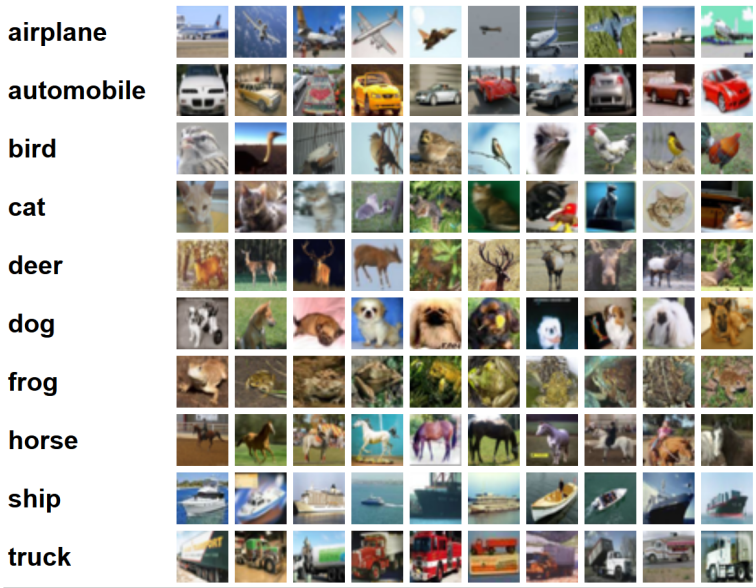

Figure 2.13. CIFAR-10 dataset examples [15].

2.3.2 Encoding

The encoding is the main difference from the MNIST dataset. The fig. 2.14 shows how the RGB colors are taken into account. The Pixel Position iM and Pixel Value iM are shared among the colors. Every color is processed separately. Only after the initial quantization, binding and accumulation, they are binded with a random HV (taken from the RGB color iM) and added together. A final binarization will produce the image HV.

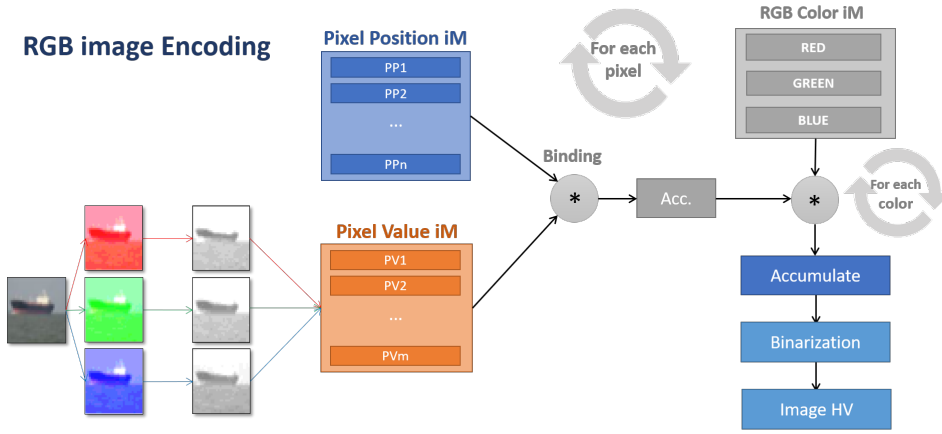

Figure 2.14. RGB image encoding. The pipeline is: preprocessing, quantization, PV and PP iMs are shared for initial binding, RGB color iM to bind and then merge the colors, final binarization.

Preprocessing

The cheapest preprocessing is the only quantization of pixel values. With respect to the grey-scale digits, the RGB images requires many more levels, from 20 up to 100. Given the initial poor results, instead of using the RGB color space we tried different alternatives (fig. 2.15):

1. YCbCr: Y is the brightness (luma), Cb is blue minus luma (B-Y) and Cr is red minus luma (R-Y). It is usually used in video and digital photography systems. In some experiments we used only the Y channel.

2. Single-Level 2-D Discrete Wavelet Transform: dwt2 matlab function which extracts horizontal, vertical and diagonal features but only if the number of pixel is increased by a factor of 10, with lower resolution no feature are extracted

3. ZCA Whitening in order to enhanced edges. For some images it works very well, if the result is thresholded under some value the noise is further reduced (look at the boat in the figure 2.15 when all the pixel value less then 0.5 are set to 0). Nevertheless, for other images the results is not so meaningful for classification.

4. Grey scale

Depending on the number of channels, the RGB color iM is either replaced by the proper one or completely removed.
An important point is that the data augmentation was never used for computational issues, but many works gain some percentage points in accuracy thanks to this trick. Data augmentation refers to a large range of techniques (such as color permutation, zoom, flips, translation) aimed to increase the dimension of the dataset.

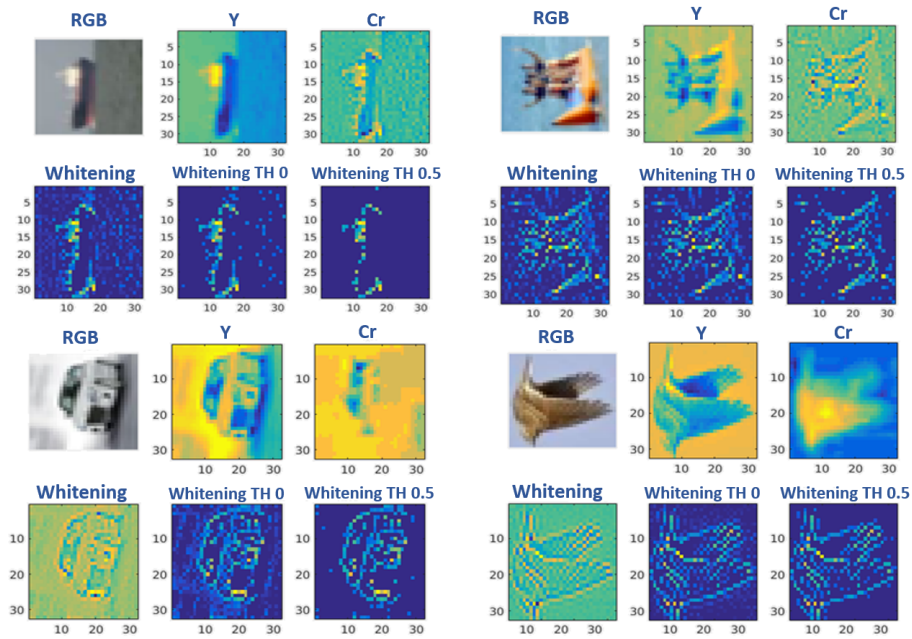

Figure 2.15. Various preprocessing techniques are shown. They work particularly well when the background is monochromatic, such as the sky. Adding a threshold to the whitening, much of the noise is removed.

2.3.3 Analysis and results

The results are very poor, even using scalar weights (fig. 2.16) and multiple prototypes per class (fig. 2.17). The maximum accuracy (35%) is reached using 5 HVs per class. It is noteworthy that the multiple prototypes let the accuracy curves grow more monotonically but it clearly overfits the training set. In figure 2.18 are compared the learning curves of grey-scale and RGB channels, the later are slightly better. Undoubtedly, the algorithm in the binary version has a strong bias due to the poor feature extraction power of our architecture. Moreover, it does not suffer of capacity issue, because increasing the dimensionality by a factor of ten (D = 100k), the results remain pretty the same, just a bit higher than expected.

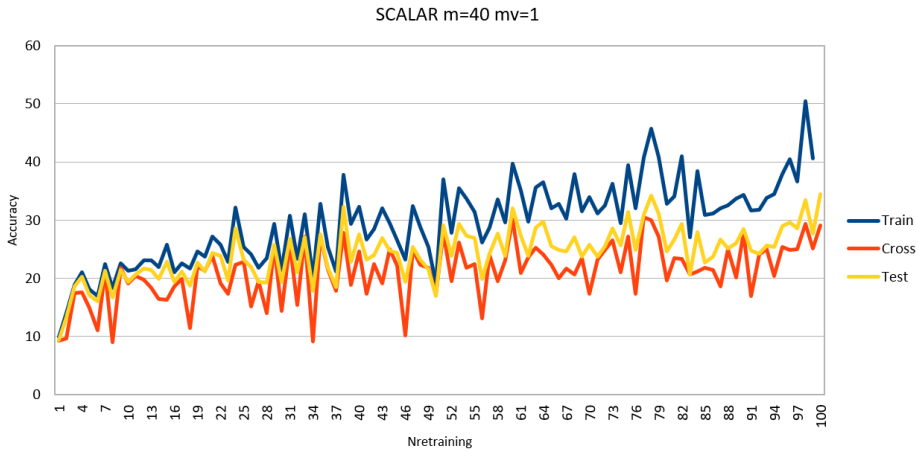

Figure 2.16. Perceptron training with scalar AM during inference, 40 levels for quantization, 1 prototype per class.

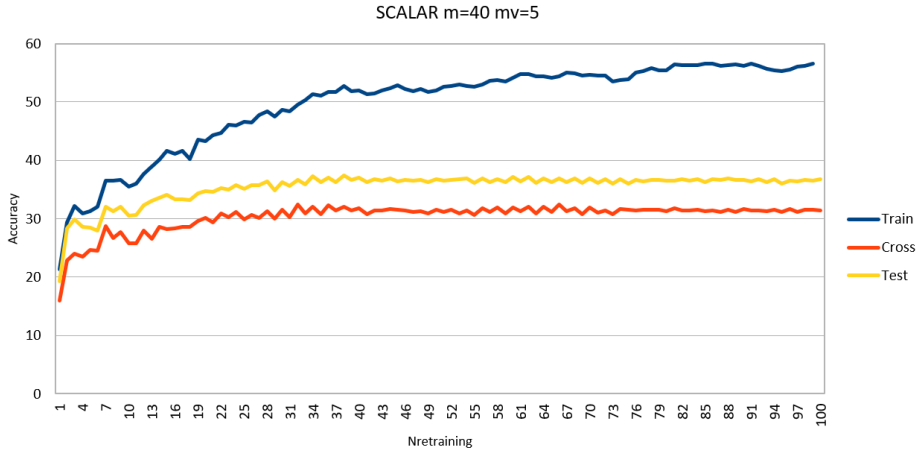

Figure 2.17. Perceptron training with scalar AM during inference, 40 levels for quantization, 5 prototype per class. Strong overfitting after the 20th epoch.

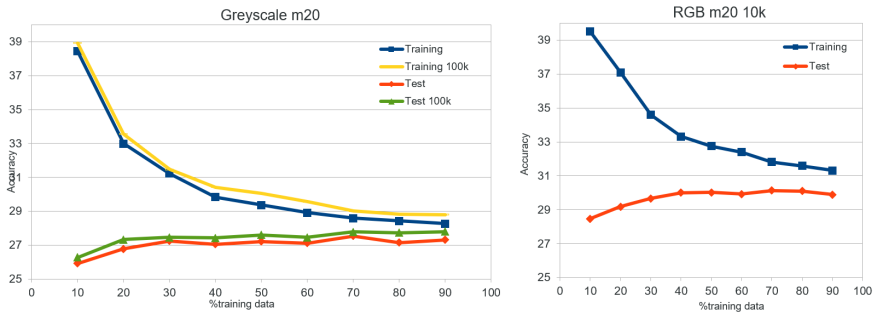

Figure 2.18. Comparison between two Learning curves with different preprocessing. They have both 20 quantization levels, but on the left side just one greyscale channel is used, on the other side the RGB which perfroms sligthly better. For the greyscale is plotted also the curves for D=100k, to check if capacity issue is present.

2.4 HD perceptron

The HD perceptron is an algorithm that we have implemented in order to adjust the class HVs learned during the one-shot learning. In fig. 2.19 is represented the overall algorithm. The training set is tested N times. For each epoch, all of the queries are compared with the Binary Associative Memory. If the guess is not equal to the label than a misprediction is met and a correction on the "scalar AM" is performed. This second AM is necessary for the back propagation of the error, just like the weights of a BNN [2], but it is never used during inference; every distance is computed in the binary domain and so with very cheap operations such as xor and accumulation. The main drawbacks with respect to the one-shot training are the memory requirements, the higher learning time and energy consumption. Indeed, it needs scalar values to be stored during the training which is repeated over the same samples more than once.

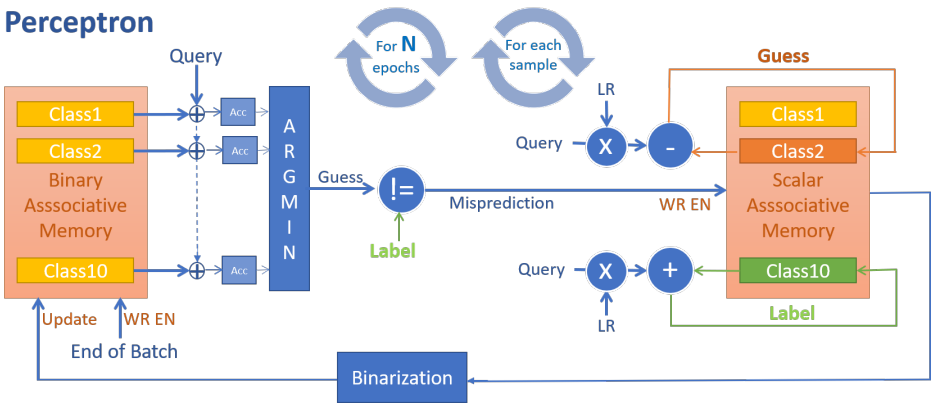

Figure 2.19. HD Perceptron training. The guess is generated by computing the Hamming distance between the Query and the Binary AM. For each mispredicted training sample an addiction and a subtraction are performed. In the example, the input label is 10, but the guess was 2, so both the Class HV are updated in the Scalar AM. The Binary AM is periodically updated by binarizing the Scalar AM.

Batch size

The Binary AM is updated after every batch with the binarized weights of the Scalar AM. The batch size have to be tuned. Contrary to a neural network, the back propagation is very fast, it requires just a sign computation, that using integer values means selecting the sign bit and get rid of all other bits. Actually this is true only for bipolar HV (-1,+1), for unipolar HV (0,1) it is necessary to take track of the number of addiction for each class in order to threshold the scalar HV with the mean value. During the perceptron, we have a random number of addiction/subtraction. This is why it is strongly suggested to use the bipolar representation and the sign function. The only difference would be in the distance computation where the dot product will substitute the Hamming distance.
The bottleneck in this case is the memory bandwidth; a typical memory with 32 bits data bus needs about 3000 access in order to update all the Class HV. For this reason, a Memory properly designed would speed up a lot the HD computing.
When the batch size is set to the maximum (the entire length of the dataset) the convergence becomes very slow. A proper trade-off should be fine in order to minimize the training time. Usually a batch size equal to 0.2% - 2% of the dataset length works well.

2.4.1 Fine grained correction

As we have already discussed, the correction is applied only in case of misprediction. It is performed by means of a subtraction and an addiction weighted by the Learning Rate (LR). The class that caused the misprediction is moved farther from the query, instead the correct one is moved closer to it.

1. For each epoch, for each mispredicted sample
   $Guess = Argmin_j(\sum_1^D AMbinary[j] \oplus queryHV)$
   $AMscalar[Guess] = AMscalar[Guess] - queryHV * LR$
   $AMscalar[Label] = AMscalar[Label] + queryHV * LR$

2. For each right guess -> no changes

3. For each batch $AMbinary = sign(AMscalar)$

The learning rate is reduced periodically during the training in order to reach a finer grained accuracy. There are two different alternatives:

1. $LR = \frac{LR}{10}$, every time the cross validation set converges, or every time a pre-determined interval is reached.
2. $LR = \frac{1}{epoch}$ or $LR = \frac{1}{\sqrt{epoch}}$: it converges fast to high accuracy, but we have less control on the back propagation and sometimes this causes instability and oscillations.

2.4.2 Dropout

The HD Perceptron is usually prone to overfitting for that kind of dataset with very dissimilar HV. For image recognition each image HV is very similar to the other in the Hamming space, so we did not use it. Nevertheless, for activity recognition it was very useful to manage the overfitting using the Dropout technique. The idea comes from the neural networks which use this kind of layer to make the architecture more generalized, preventing complex co-adaptations [16]. In the normal case, half of the features are randomly omitted, for our purposes half of the bits are usually not enough; common values are between 0.7% and 0.9%. One naive way for setting the number of masked bits is to increase them until the accuracy of the training set cannot reach 100%, or it can reach it only after many epochs.

During the HD Perceptron training, a mask is applied to each input query (fig. 2.20). This mask is changed at the end of every epoch. The masked query follows the same pipeline of the architecture without any dropout: so both inference and scalar AM update are performed using the masked input. In this way, for each epoch, the training focus on different features and the overfitting is reduced.

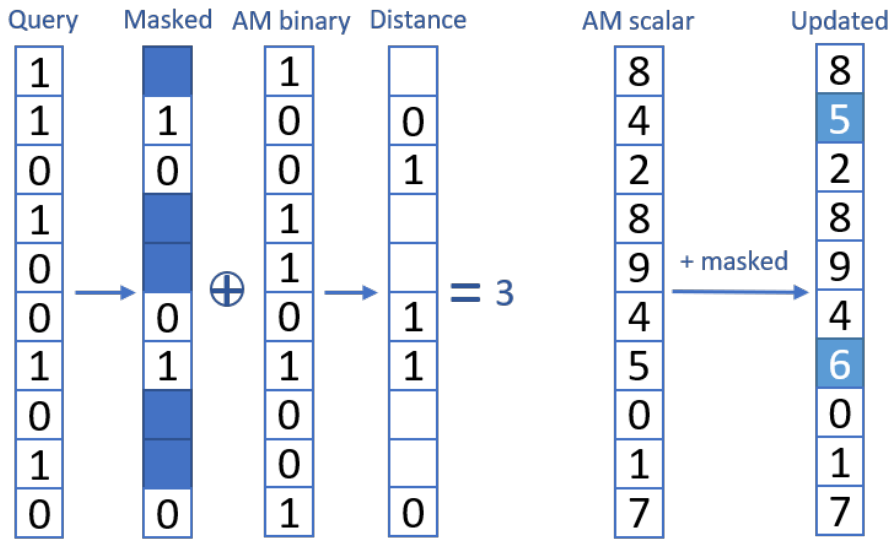

Figure 2.20. The dropout mask applied during inference and back propagation.

2.5 Positional binding

The item Memory for encoding the position of the pixel was initially generated randomly, filled with orthogonal HVs. In the images, there is a strong correlation between the position of each pixel. Taking this fact into account, the architecture becomes almost invariant from slight translations and rotations.
A simple continuous iM is not enough because it is suitable for vectors, instead we need two dimensions to define a Positional Binding Matrix. Starting from any pixel of the image, the associated pixel position HV (PP) is very closed to the PPs refered to adjacent pixels. As soon as we move away from this pixel, the distance will increase. The maximum distance is obtained comparing the PPs of two opposite corners. In figure 2.21 some examples are shown.

In order to obtain the Positional item Memory (PiM) we have to combine two CiM in a proper way [17]. For 1D signal that is discretized into m levels, we simply choose a random hypervector for the minimum level and randomly flip $\frac{D}{2*(m-1)}$ of

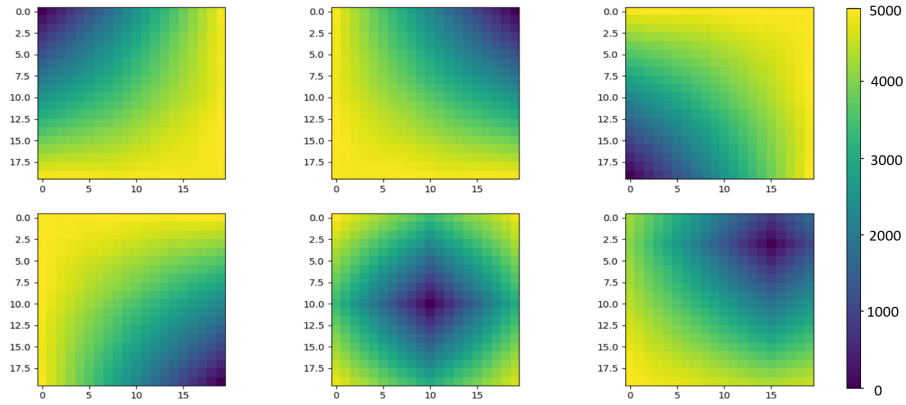

Figure 2.21. Each plot shows the distances of each PP with respect to the dark blue pixel (Hamming distance = 0). Moving away from this point the color changes until it becomes totally yellow, which represents orthogonality (Hamming distance = D/2). Regardless of the initial point, the behaviour is always the same.

its bits for each successively higher level (once flipped, a bit will not be flipped back). In this way a CiM is formed. It is much easier to flip rather than to keep a MIN and MAX value as explained in the paper. The final result is the same and we have to store only two random vectors (MIN level for X and Min level for Y) instead of four (MIN, MAX for both X and Y).

This kind of item memory can be useful for any future work which is based on HD computing and needs to encode the position on the input. For instance to encode the position of different digits or object in a window [18] or more generally for Visual Scene Understanding [6], the application would be straightforward.

2.5.1 Demonstration

Each location is obtained binding the value of x,y axis, where each of them belong to two different CiM. So we encode a scalar value x to a vector $X = f(x)$. Similarly, we encode the scalar value y, to a vector $Y = g(y)$; I use g() since it uses orthogonal seeds with respect to f(). The location L would be:
$L = loc(x, y) = f(x) * g(y) = X * Y$

We have to find the exact Hamming distance between two pair points, or at least bound the distance below a value. We know that:

$$h(X_i, X_j) = \frac{D}{2 * n_x} * abs(j - i) = Cx_{ij} \tag{2.1}$$

for each (i, j) in $[1 : n_x]$, where n_x is the number of pixel on the x axis and $h()$ is the Hamming distance function. In the same manner:

$$h(Y_k, Y_l) = \frac{D}{2 * n_y} * abs(k - l) = Cy_{kl}$$

for each (k, l) in $[1 : n_y]$. Moreover X_1 and Y_1 are generated orthogonal and so also X_i and Y_j would remain orthogonal. In fact flipping randomly n bits form two random hypervectors the result is still orthogonal because of the equal density of 0s and 1s. $h(X_i, Y_j) = \frac{D}{2}$ for each (i, j) .
We can compute the distance between every couple of pixels $h(X_i * Y_k, X_j * Y_l)$ using a probabilistic approach. First of all, the Hamming distance between unipolar HVs is computed with a component-wise xor. Besides each factor is the binding of two unipolar HVs that is obtained with an xor again. For the commutative principle, it is valid the following equality:

$$h(X_i \oplus Y_k, X_j \oplus Y_l) = |X_i \oplus Y_k \oplus X_j \oplus Y_l| = h(X_i \oplus X_j, Y_k \oplus Y_l)$$

where $||$ counts the number of ones in the vector.
Using the aforementioned formula we can compute the number of ones and zeros of $X_i \oplus X_j$:

$$\#1s = Cx_{ij}$$

$$\#0s = D - Cx_{ij}$$

In the same way $Y_k \oplus Y_l$ has Cy_{kl} 1s and $D - Cy_{kl}$ 0s.

Given a generic feature n between $[1 : D]$, the probability to get a "1" computing $X_i[n] \oplus X_j[n] \oplus Y_k[n] \oplus Y_l[n]$ is: $P(X_i[n] \oplus X_j[n] \oplus Y_k[n] \oplus Y_l[n] = 1) = P(X_i[n] \oplus X_j[n] = 1) * P(Y_k[n] \oplus Y_l[n] = 0) + P(X_i[n] \oplus X_j[n] = 0) * P(Y_k[n] \oplus Y_l[n] = 1)$. More concisely:

$$P(xxyy = 1) = P(xx = 1) * P(yy = 0) + P(xx = 0) * P(yy = 1)$$

where the addens are all possible cases to get a 1 having an xor.

$$\begin{aligned} P(X_i[n] \oplus X_j[n] = 1) &= \frac{Cx_{ij}}{D} \\ P(X_i[n] \oplus X_j[n] = 0) &= D - \frac{Cx_{ij}}{D} \\ P(Y_k[n] \oplus Y_l[n] = 1) &= \frac{Cy_{kl}}{D} \\ P(Y_k[n] \oplus Y_l[n] = 1) &= D - \frac{Cy_{kl}}{D} \end{aligned}$$

$$P(xxyy = 1) = \frac{Cx_{ij}}{D} * \frac{(D - Cy_{kl})}{D} + \frac{(D - Cx_{ij})}{D} * \frac{Cy_{kl}}{D} = \frac{(C_x + C_y)}{D} - \frac{2 *_C x * C_y}{D^2}$$

Finally:

$$h(X_i \oplus Y_k, X_j \oplus Y_l) = P(xxyy = 1) * D = (C_x + C_y) - \frac{2 * C_x * C_y}{D} \tag{2.2}$$

The fig. 2.22 shows the differences between the ideal case (computed with equation 2.2) and the real case (using HVs) where the distances are pretty different because the 1s and 0s in a random hypervector are not exactly identically distributed in each zone of the HV. I used a matrix $n_x * n_y = 10 * 10$ just for representation purposes.

I would like to underline that using the formula 2.1, e.g. nx = 10 and D = 10000, we obtain $h(X1, X10) = 4500$ and not 5000. In order to get it we have to

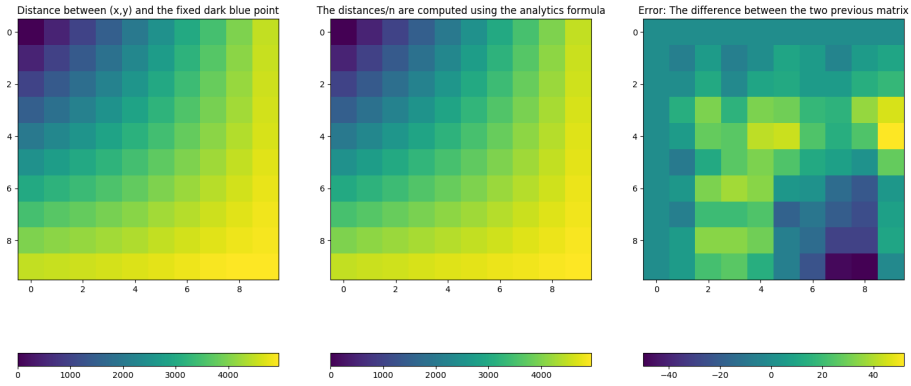

Figure 2.22. The real PiM is quite identical to the one computed with the analytic formula. The last plot shows the error between the two previous cases. $\epsilon \in (-50, +50)$. Having D=10000 the relative error is just 0.5%, and 10% wrt the distance between two adjacent HV Cx_{ij}

change the formula in: $h(X_i, X_j) = \frac{D}{2*(n_x-1)} * abs(j-i)$. However, it gives slightly worst results moving the Pearson correlation from 0.97 to 0.96. Using eq. 2.1 (X_1, Y_1) would be orthogonal only vs (X_{10}, Y_{10}). Instead with the other formula it would be orthogonal vs (X_{10}, Y_{10}) but also vs $(X_{10}, :)$ and $(:, Y_{10})$, so to all the pixels position HV belonging to the opposite edges.
The Pearson coefficient between L1 and Ham is 0.97 (fig. 2.23); using Norm2 instead of L1 to calculate the distance between (i,j) and (k,l) I got 0.975. The shapes are not exactly linear because of the distances grew along orthogonal axis and not ideally in a spherical way.

2.5.2 Sandwich iM

If we are interested in a strict local spatial correlation, the Sandwich iM is the proper choice. In one dimension it is a kind of CiM where each point has a Hamming Distance equal to $\frac{3}{4} * D$ from each neighbour, and it is orthogonal to all the others. It is easily computed by initializing the HV in odd positions to random HV, after that the even HV are computed coping half of the bits from the neighbor on the left size and the remaining bits from the right size. Scaling it to the two dimensional

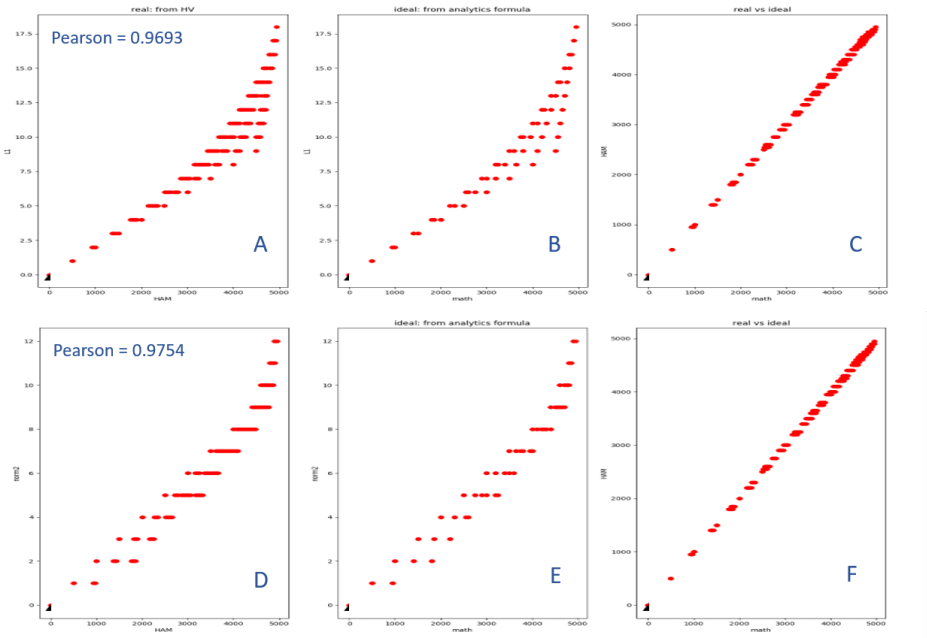

Figure 2.23. L1 distance (A,B) and Norm2 distance (D,E) vs Hamming distance between couple of coordinates (x,y) in the real and analytic case respectively; the error between them is scattered in (C,F).

case, we can easily repeat the 1-D Sandwich iM for multiple rows (fig. 2.24) but in this way just horizontal spatial correlation is obtained.
To force correlation in all the 8 cell neighbour, the algorithm is more complex.

1. odd Vertical (V), odd Horizontal (H) : random HV
2. odd V, even H : half from the HV above, half from the under one
3. even V, odd H : half from the HV on the left, half from the HV on the right
4. even V, even H: a quarter from each pixel on the four corners of the neighbor.

The result (fig. 2.25) is not exactly ideal, but usually it does not hurt. It is the only way I have found to keep in consideration the diagonals in the neighbour.

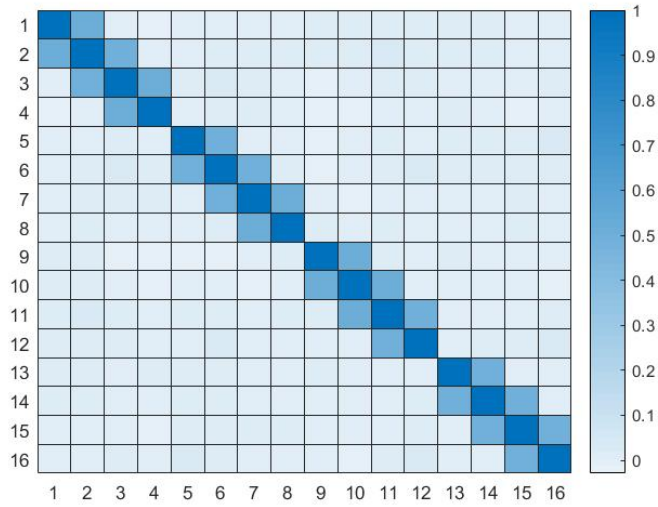

Figure 2.24. Intra-pixel Cosine similarity of a 4x4 matrix. Each pixel has a distance of about 7500 from the following and the previous pixel, except the pixel on the boundary. It is clear that there are 4 blocks and each one refers to a row.

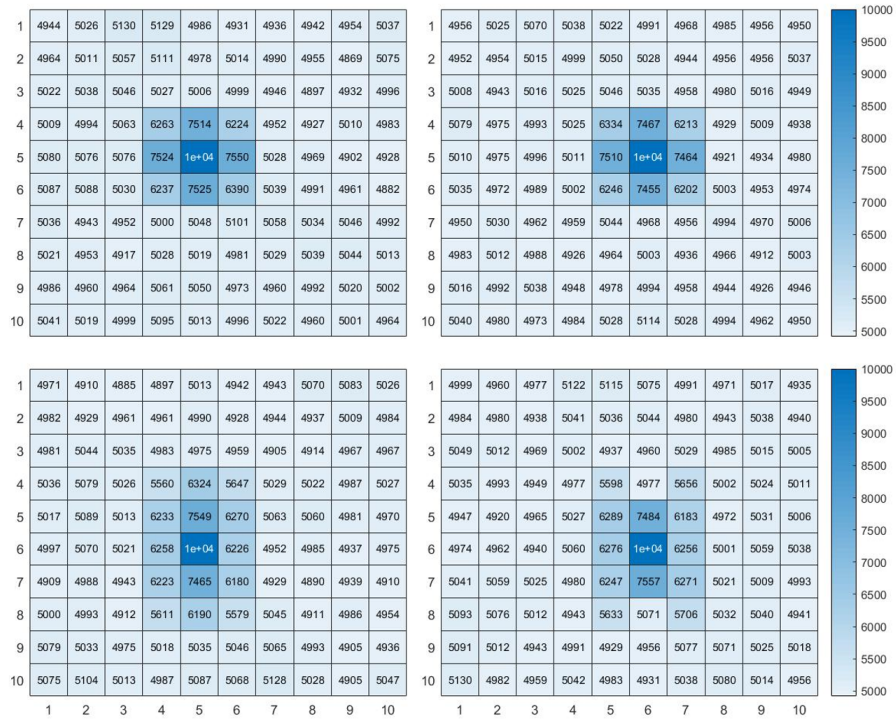

Figure 2.25. The ideal case is obtained in the upper half plots, where are plotted all the Hamming distances from a single point placed in an odd row of a 10x10 matrix. In the even rows the similarity in the neighbor is maintained but it is larger and not totally ideal.

2.6 Multi-prototypes

As already discussed, storing more than one HV for each class in the AM will produce many advantages at the cost of memory. First of all, how is the classification carried out? We have considered two alternatives:

1. Maximum metric: the query is associated to the class which contains the most similar prototype.

2. Majority vote: the prototypes are ranked depending on the distance. The top "n" ones vote for selecting the class. The n parameter is usually chosen small and for sure less than the number of prototypes per class. In our test we set n=3, in case of ties (one vote for each class) the closest one is chosen.

The maximum metric enhances the outliers, the majority instead gets rid of them if the other votes belong to the same class. This is the fundamental idea of kNN algorithm. Ideally, each prototype represents a cluster. However, if the cluster is composed by few samples and it is placed very close to other clusters belonging to different classes, it would cause a huge number of mispredictions.

2.6.1 Training

Many alternatives are available for the training. It is possible to take a clue from already existing clustering algorithm but usually they are too expensive. A naive idea is to compute repeatedly the training and each mispredicted sample is added to a new prototype which is changed every epoch. To make it clear, all the samples of class x mispredicted during epoch "n" will generate the prototype number "n+1" of the correspondent class. This method does not imply a good cluster subdivision, but it excludes from the new prototypes the most recurrent samples, fixing the issue explained in the previous section 2.2.3 that is probably the main problem for the digit recognition.

Chapter 3

Kmeans + HD

There is a good amount of work using unsupervised nature of clustering methods which can be exploited as back-end feature extractor and combined with HD operations as the front-end [3, 19, 20]. If we divide the input images in small squares (patches) of few pixels, we can apply some filters to extract meaningful information from the patches: each filter will produce a feature. Having many filters we will generate an HV of features. These filters are learned using a clustering method, each computed centroid in the patches space will be used as a filter which populates a huge Dictionary. The clustering will increase a lot the training time, but during inference, it will require just a matrix multiplication between each patch and the learned Dictionary. So we are slightly increasing the complexity, in order to drastically improve the accuracy (up to 70%).

3.1 Background

The clustering methods alone (with large number of centroids 1600-4000) showed 70-80% accuracy on CIFAR-10 [3], on par with the ARM CMSIS-NN that showed 80% accuracy with an inference speed of 10 images/s in a Cortex-M7 core [21]. Our aim is to reach quite the same performance of ARM architecture but using faster and cheaper computations.

3.1.1 Kmeans

"The K-means algorithm clusters the data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares." [22]

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (\|x_j - \mu_i\|^2)$$

I will not explain the algorithm in details, but only some interesting aspect for our application. For further details, refer to the aforementioned scikit-learn library from which we have selected the clustering algorithm.
The main parameter to be set is the number of centroids. They are randomly initialized (or associated to some input patch randomly selected) and then iteratively moved toward the direction which minimizes the inertia. At each iteration, the samples are re-associated to the nearest cluster. The aim is to find a good number of clusters which may represent meaningful features for image classification; for this reason clusters with only one elements have to be avoided.
The kmeans++ initialization is often suggested in order to speed up the convergence. We observed that as soon as the number of samples increase up to the order of millions, this kind of initialization would become slower than the entire training; so we decided to remove it.

Spherical K-means

Spherical K-means is a modified version of the original algorithm aim to find a Dictionary D and a new representation (a code vector $s^{(i)}$) of the input ($x^{(i)}$) which satisfied the following criteria [21]:

$$\min_{D,s} \sum_{i=0}^{n} (\|Ds^{(i)} - x^{(i)}\|_2^2)$$

$$\|s^{(i)}\|_0 \leq 1, \forall i$$

$$\|D^{(j)}\|_2 = 1, \forall j$$

Alternating the optimization of D and "s" is possible to compute a very large Dictionary very speedily.

3.1.2 Whitening

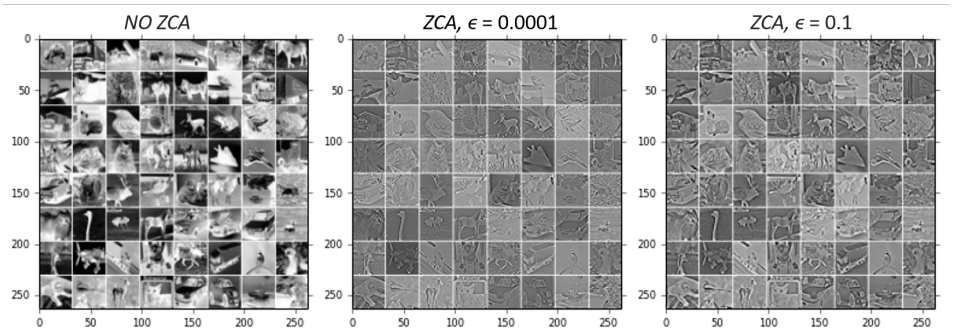

Figure 3.1. Some example of ZCA Whitening application on CIFAR-10 dataset. $\epsilon = 0.1$ seems to be optimal because the edges are clearly underlined, instead $\epsilon = 0.0001$ is too low, indeed the high frequency noise is amplified.

In Literature, many works on computer vision make use of some preprocessing technique before training. One of the most common is the whitening: a particular transformation which produces an identity covariance matrix, hence all dimensions are statistically independent and the variance of the data along each of the dimensions is equal to one. The unitary variance make all dimensions equally important, the independence instead has many probabilistic advantages. There are two types of whitening: PCA and ZCA [23]. The ZCA is the most suitable for images because it stretches the dataset to make it spherical, but tries not to rotate it, whereas PCA does rotate it a lot.
For vision data, high frequency data will typically reside in the space spanned by the lower Eigenvalues. Hence ZCA is a way to strengthen these, leading to more visible edges. The whitened points are computed as:

$$W = V(D + \epsilon_{zca} I)^{-1/2} V^T x$$

where ϵ_{zca} is a small constant. The correct tuning of this constant is important, a too low value would amplify high frequency noise, making more difficult the classification. In the paper [24] $\epsilon_{zca} = 0.1$ is suggested for 8x8 pixel patches, lower

values for smaller patch sizes (fig. 3.1. In order to further understand how whitening works, we can plot the Whitening Matrix as shown in [25]. Each row of W is a kind of local filter which is estimated from the patches. The filter in row N is an impulse, surrounded by inhibitory weights, centered on the Nth pixel of the image (fig. 3.2) (counting pixels starting in the upper left corner, proceeding down columns).

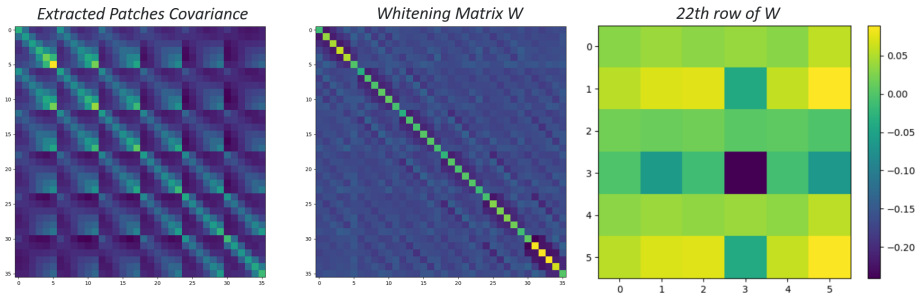

Figure 3.2. Each row of the whitening Matrix (center) represent a filter (right) which is applied locally as an impulse surrounded by inhibitory weights.

3.2 Method

The following architecture replicates the one introduced in [3], further details on the k-means algorithm and the whitening procedure are given in [24]. The only differences lie in the HD front-end and the parts concerning the binarization. Our proposed architecture will have three modules:

1. In the back-end, we use a clustering method to map an image patch to an intermediate real vector with a larger dimension (equal to the number of centroids). Then we transform the real vector to a binary D-bits (D to be defined) vector representing the patch (let's call it patch vector).

2. We use a positional binding (sec. 2.5) with HD operations to bind the patch

vector computed from the back-end with a "location vector" encoding the location of the patch in the image. The location vector is another D-bits binary vector repressing the relative 2D location of the patch inside the 32x32 image (can be easily computed on the fly from x,y coordinates of image patch in 2D space). Essentially, here, we want positional binding to be robust with respect to small changes in location, so that if we change feature positions by a small amount, then this results in a small change to the resulting image representation. This would pave the way for translation stability [17]. For every patch, we bind (XOR) its location vector to its patch vector. Then, we pool (add by means of majority rule) these bound vectors across an image quadrant. Concatenate the D-bits binary vectors of each quadrant to construct the final 4*D-bits binary vector representing the entire of image.

3. After image encoding, in the front-end, we construct a/few prototype(s) per class based on the computed image vectors from the previous step.

Overall, (2) and (3) are trivial with the well-known methods but (1) is challenging. We need to find an efficient way of transforming an image patch to a large binary code that can be bound to the binary code of patch's location. In sec. 3.4 are proposed two methods: thermometer code and grey code.

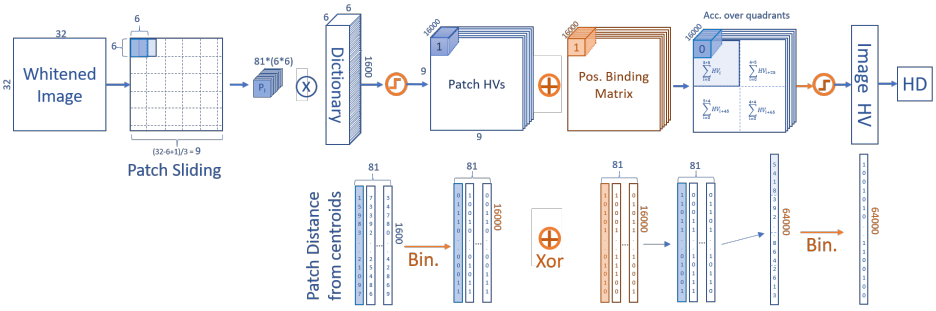

Figure 3.3. Kmeans+HD Pipeline to generate the image HV: preprocessing, patch sliding, Feature mapping, quantization and binarization, Positional binding, accumulation over quadrants, final binarization.

The main points of the pipeline are shown in the fig. 3.3. Let's describe them in details:

1. Preprocessing: Normalization and Whitening of RGB images

2. Patches computation: each patch has $w * w$ dimension (the paper reports w=6 as the optimal parameter), but having three color channel the total number of features per patch is 108 = 6*6*3 (in the image just one channel is represented). The number of patches per image depends on the stride, which is the pixel step between two following patches; best stride values are between 1 and 3. $\#Patches/image = (floor(\frac{32-w}{stride}) + 1)^2 = 729, 196, 81$ respectively for stride = 1, 2, 3.

3. Dictionary computation: having 50000 images for training, I have some millions of patches. I select randomly 1 million of them as input of a k-means Clustering function. This will produce "K" centroids, we have to find the optimal trade off between memory-power requirements and the accuracy (that is proportional to #centroids), good candidates are K = 1000, 1600, 2500.

4. Feature mapping: the "Triangular" activation function was used, as specified in the paper:

$$f_k(x) = max\{0, \mu(z) - z_k\}$$

where $z_k = \|x - c(k)\|_2$, "c" is the vector of centroids and $\mu(z)$ is the mean of the target patch 'z'. Another possibility is to perform a simple matrix multiplication without any activation, $f(x) = z * c \rightarrow$ 1*108 x 108*2500 $\rightarrow \mathbb{R}^K$ e.g. $\mathbb{R}^{2500}$. Indeed, the activation can be removed because of the following quantization and binarization step performs a similar thresholding operation.

5. Rescaling: moving from the space $\mathbb{R}^{108}$ to $\mathbb{R}^{2500}$, each component (feature) is the L1 distance of the patch from a centroid. Even if the features are all distances, and because of this they should have a comparable scale, I found that normalizing feature by feature over all the patches is necessary (10% accuracy gain). With normalization I mean : $\frac{mean(i)-feature(i)}{VAR(i)}, \forall$ feature i. Probably this rescaling helps a lot the quantization step because the variance of the data is reduced.

6. Quantization $\mathbb{R}^K \rightarrow \mathbb{N}^K$: given our results, it requires at least "q" = 6

number of quantized levels to maintain quite the same distances between patches in $\mathbb{R}$ space.

7. Binarization $\mathbb{N}^K \rightarrow \mathbb{H}^D$: we explored three possible alternatives: thermometer code, gray code, random projection (sec. 3.4).

8. Generation of a PBv = Positional Binding $\mathbb{H}^D$ vector for each patch position. The Hamming distance between the PBv of patch in position (i,j) and PBv2 in position (k,l) is quite linearly proportional to $\|[i,j],[k,l]\|_2$. We got both empirical results and theoretical demonstration in sec. 2.5.

9. Pooling: the image is divided in four quadrants; all patches belonging to each quadrant are added together to reduce dimensionality of the HV.

10. Final Binarization to get the image binary HV used during training.

In the first experiment we used SVM algorithm as front-end in order to check whether we can get the same results reported in that paper using a quite identical architecture. The only differences are that they do not use (5,6,7,8) but we need them in order to finally substitute the SVM with the HD computing and so getting a really fast binary inference.

3.3 Centroids generation

The main issue for the centroids generation was to find the best number of patches which generates the best Dictionary (which gives higher classification accuracy). Fixing the stride=1, there are 729 patches available per images: using the full dataset and only 1 million of patches randomly chosen to train the dictionary, I am using in practice only 3%*729 = 22 patch per image. Such a small number of patches per image could not be able to extract the useful features, for instance choosing just the patches targeting the sky and the ground.
The number of patches per images is a parameter that is not exploited in [3]. Let's call this parameter P. In another paper of Coates [24] they talk about the right number of inputs to get a good Dictionary using K-means (where centroids are sparse directions in the input data), but they do not talk about P. It is reported: "These patches can be collected from unlabeled imagery by cropping out random

16-by-16 chunks. In order to build a complete dictionary, we should ensure that there will be enough patches so that each cluster can claim a reasonable number of inputs. For 16-by-16 gray patches, m = 100k patches is enough. In practice, we will often need more data to train a K-means dictionary than is necessary for other algorithms (e.g., sparse coding), since each data point contributes to just 1 centroid in the end."
As of my results, with 6x6 patches I gained the following results, which agree with the paper:

1. more patches → more sparsity → less number of centroids with only one element → less overfitting
2. 25k patches, 1600 centroids → 636 centroids with more than one element
3. 100k patches, 1600 centroids → 1179 centroids with more than one element

So what about P? A small value of P could be fine for a large dataset. The most important thing is to use enough large number of Patches but 1 million could be fine. There would be many patches included in only a small set of centroids (for instance that one targeting a piece of sky), but it would remain a good percentage that covers the meaningful features.

3.4 Binarization

Considering an image patch of 6x6, the transformation does $\mathbb{R}^{36} \rightarrow \mathbb{R}^d \rightarrow \mathbb{H}^D$, where $\mathbb{R}$ is the real space, $\mathbb{H}$={0,1}, D > d, and D≅10K. This binary embedding for image patches should meet the following conditions:

1. The embedding should preserve the L1 distance between points in the $\mathbb{R}^d$ space to Hamming
2. The produced binary code should have a mean of 0.5 (equal number of 1s and 0s) to be able to use it for positional binding with XOR.

3.4.1 Thermometer code

There are distance-preserving binarization methods to meet the first condition, e.g. bit sampling locality-sensitive hashing (LSH) [26, 27], but their binarized code do not have equal number of 1s, and 0s hence violates the mean 0.5. Here is one idea to go around the non-0.5 mean issue with bit sampling LSH.
Let's use clustering to learn K centroids for image patches (K=1,000). Then, we use bit sampling LSH to map $\mathbb{R}^d \rightarrow \mathbb{H}^D$, where $D = c * K$, and c is the largest value of the centroids (e.g., $c = 10$ if we scale/quantize the range). Essentially, every float value of a centroid is represented by c bits using unary coding (or thermometer code): a value of v is represented by v 1s followed by $(v-c)$ 0s. The D-bit binary code, b, is constructed by concatenating d unary codes, each with c bits. It is quite clear that generated b code by cannot meet the mean 0.5 but preserves exactly the distances. For instance (137,248,47,450) is re-scaled to (1,2,0,4) and then binarized in 4 codes of c=4 bits (1000 - 1100 - 0000 - 1111). The Hamming distance between 1111 and 1000 (4 and 1) is clearly 3 (4-1) by construction, so it maintains exactly the distance between integer values.

To make the mean of b vector 0.5, we can simply "randomize" it by multiplying (XORing) it with a random vector: $L * b$. L is a D-bit HV with equal number of 1s and 0s, randomly chosen (we only need one L for the entire of learning and classification task). When we map two vectors, e.g., b1 and b2, this simple coding maintains their temperature differences because XOR preserves the distance so does not affects Hamming distance h: $h(b1, b2) = h(L * b1, L * b2)$
Essentially, it is a powerful way of encoding graded values (quantities) into binary vectors via the thermometer code since neither XOR nor permutation affects Hamming distance. It requires q bits per q quantization levels, but the unipolar vector $\mathbb{H}^{c*K}$ maintains exactly the same distances of $\mathbb{N}^K$; so the only distortion between distances is produced by the quantization step (fig. 3.4). Improving the quantization, by selecting optimal intervals looking at the sample distribution, would increase the performances.

Bit distribution

Even if the mean of each code is 0.5 (or 0 in the bipolar version), they are not randomly distributed. For instance, the first bit of each code is likely to be one independently on the quantized level, in the same way the last bit is 0 in most of the case. Using a c-bit code for each interval, the probability (P[i]) of each bit to be 1 decrease as soon as we move forward to the last bit: $P = \{\frac{c-1}{c}, \frac{c-2}{c}, ...\frac{2}{c}, \frac{1}{c}\}$. This problem could be fixed by using both the complementary thermometer codes. So the interval 1 is represented as 10000 or 01111 (complementary version). Half of the values are assigned to the first code, the other half to the complemented one. In this way, adding many codes together during the pooling step, it will produce an HV where each bit is not biased and so it has the same probability to be 0 or 1. The only drawback is that the complementary thermometer version will map the inputs in a different zone of the HD space. This means that the dataset is virtually divided in two parts and each subset is trained separately.

3.4.2 Gray code

On the other hand, we may use ad hoc mean-0.5-preserving binarization methods, but they often violate the distance preserving. One could assign c-bit to a centroid value by linearly map the value to one of possible M levels. We arrange this mapping LUT with M rows to construct a "gray code" where every two successive mapped values differ in only two bits (e.g. q=4 0011,0101,0110,1010,1001,1100). But there is no mathematical guarantee, just empirical tests. It does not maintain exactly the same distance in $\mathbb{N}^K$ but the number of available levels is much higher $\binom{q}{q/2} = \frac{q!}{\frac{q}{2}!*\frac{q}{2}!}$: 6, 20, 70, 252 levels using respectively q = 4, 6, 8, 10. In order to evaluate the goodness of these space transformation we used the Pearson correlation coefficient $\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$, where cov() means covariance and σ is the standard deviation. D = dimension of our binary vector will depend on "K" and "q" choices: D = K*q. From the third row of fig. 3.4 it is clear that the smaller the L1 distance, the better it is preserved. This behavior can be useful for our purposes, because we mainly care for identifying which patches are actually similar to the query, for all the others it is sufficient to know that they are distant from the query even if the computation of this distance has low precision.

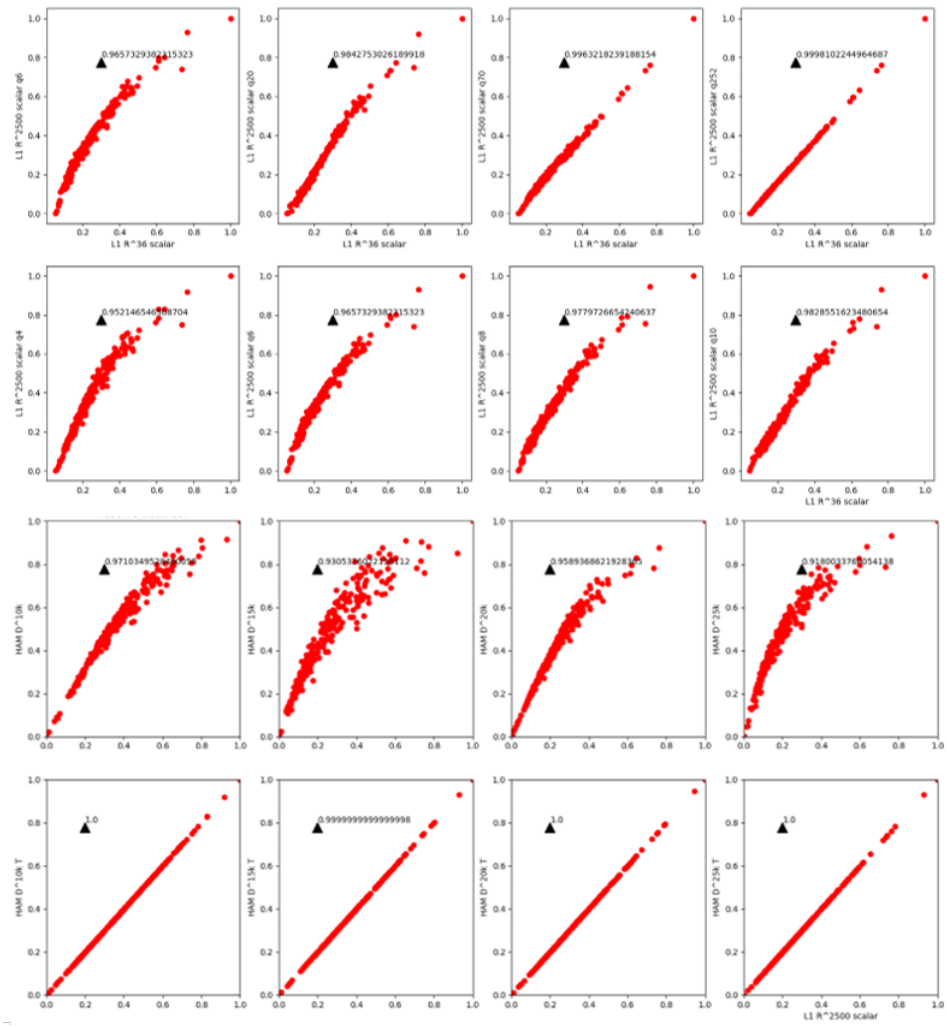

Figure 3.4. The distortions cause by quantization and binarization are shown. 1st row quantization error for Gray code levels (6,20,70,252). 2nd row: quantization error for Thermometer levels (4,6,8,10). These levels was properly chosen to compare Gray and Thermometer codes with teh same number of bits (D*4, D*6, D*8, D*10). The 3rd and 4th rows shows the binarization error due to Gray and Thermometer code respectively. The former causes high distortion when the distance is larger, the latter has no loss instead.

3.4.3 Random Projection

The last technique we propose is the random projection. There are many articles and theorems which carefully described the properties of this method [28]. In particular there is The Johnson-Lindenstrauss Lemma which states: "given a set S of points in $\mathbb{R}^n$, if we perform an orthogonal projection of those points onto a random d-dimensional subspace, then $d = O(\frac{1}{\gamma^2} \log |S|)$ is sufficient so that with high probability all pairwise distances are preserved up to $1 \pm \gamma$".
So we can reduce the dimensionality of the input from n to d, preserving the distances with a maximum error that is fixed by the d choice. If the input signal in $\mathbb{R}^n$ have an high similarity, then the admitted error should be very low and so d very high. However all those considerations are valid in the real space. Our aim is not dimensionality reduction, but binarization. There is not any theorem which limits the error in the hamming space, so we apply the random projection in a experimental way. However it is reasonable to consider the binarization error proportional to that one computed using the Lemma. Setting d = n is comparable to a 64-32x dimensionality reduction and usually it is a too high factor for distance preserving; so for binarization d is usually higher than n, but the d are bits not scalar values.
Each row P_i of our random projection matrix P is randomly generated from a Gaussian distribution. P_i can be considered as an axis in the $\mathbb{R}^n$ space; every input point X is projected on this axis, then only the sign is taken $s = \sigma(P * X)$ (where sigma represent the sign operation and * the dot product). So, each axis will produce a bit which is 1 if the input is placed over this axis, 0 otherwise.
We have already seen that the thermometer code preserves the exact distances, moving from integer to binary, but it requires a huge number of bits, equal to the maximum integer value. In order to limit the quantization loss, this value to be as greater as possible (good results are obtained from 8-10 levels). Hence the random projection turn to be useful if d < 8*n.

Distance metrics

The Johnson-Lindenstrauss Lemma guarantees to maintain the same L2 distance, let's see if using Cosine Similarity would affect the distances. Given the query x and an HV y in the Associative memory:

$$\|x-y\|_2 = (x-y)^T*(x-y) = x^T*x-2*x^T*y+y^T*y = x^T*x+y^T*y-2*cos(x,y)$$

so if $x^T * x$ and $y^T * y$ are constant the metrics L2 and Cosine Similarity are proportional and the distances are maintained after the Random Projection. In our data $y^T * y$ is not constant for each y in the AM. If I normalize the AM, the L2 distance will give the same results than cosine similarity.
The dot product works fine as well. Indeed, the dot product and the cosine similarity give the same results if the data are normalized or binary, because in both the cases the norm is constant for all the samples and so $\cos(x,y) = dot(x,y)/(norm(x)*norm(y))$ where the denominator is just a scaling factor.

3.5 Experiments

In this section, we will describe alternative architectures that we have implemented to improve the accuracy of the algorithm. Unfortunately, no one have played a particular role at really improving the performance. The 10% gap in accuracy between scalar and binary version remains open. Nevertheless, they are all interesting algorithm that can be used in future architectures based on HD.

3.5.1 RP as Dictionary

In this case, the Random Projection (RP) was used as feature extractor instead of dimensionality reduction. We replaced the Dictionary, composed by K centroids, with a RP matrix in the $\mathbb{R}^{108xN}$ space, where N was chosen equal to the number of centroids. The projection has a similar effect to the k-means centroids distance computation. They both compare each patch to a dictionary of points of the space, but the k-means algorithm compute these points based on the patches distribution, instead the RP just keep them orthogonal. The results are comparable to the k-means architecture, just a slightly loss in accuracy. The main advantage is the

training time, the k-means is in fact the bottleneck of the training computation. The inference speed remains the same and the memory requirements as well. A maximum accuracy of 52.8% was reached in the fully binary architecture trained by the svm. Removing the last binarization step, the accuracy grows up to 63%, which is comparable with 64.5%, obtained with the k-means instead of the random projection.

3.5.2 Hard k-means

Starting from the idea that the nearest centroids are the most representative ones of the patch, instead of keeping in consideration all the distances from all the centroids, we can merely pick the closest one(s). For each centroid there will be an orthogonal D-bit vectors in the item memory (D= 10K). So, each patch will be mapped to only one(some) vector of the item memory. Then positional binding is applied, and it follows the usual pipeline. In this way, the image HV will store a histogram of centroid appearances. The result was poor, probably because this algorithm is strongly affected from the initial centroid selection.

3.5.3 HD clustering

As we have already seen in the previous chapter, the multi-prototypes technique has a great number of advantages, but the selection of the right number of prototypes is a tricky task. The HD-clustering is a method to define this number. I have transformed the k-means algorithm into the HD space in a naive way and I used it to compute the clusters. It is divided into two main steps repeated until no further cluster changes during update:

1. Assignment: each sample is assigned to the cluster with minimum Hamming distance.

2. Update: each cluster is updated as the mean (majority) of all the samples belonging to it.

After the unsupervised training, I defined the prototypes by labeling each cluster using the majority rule. For the initialization random samples was used.

The training accuracy (computed after the cluster labelling) increases with the number of clusters, but it is quite low: 30% with 50 clusters, 40% with 400 and 50% using 5000 clusters. I tried multiple random initialization using 50 clusters and the result changed but not too much (±2%). However, I left this method because the algorithm was converging to too poor testing accuracies and so this gave us not so many useful information about the right number of prototypes.

3.5.4 Dynamic HD perceptron

Another algorithm that I have implemented is a kind of HD Perceptron where I dynamically increase the number of prototypes each time I get a misprediction. I have fixed the maximum number of prototypes to 200, when the AM becomes full then the next new prototypes will substitute the most useless one. The most useless is identified by a counter that is increased each time the target prototypes is the most similar to the query, and so when the target prototype produces the final guess which will be used for classification. Finally, I selected a subset of prototypes with the highest precision that means $\frac{\text{correct guesses}}{\text{total guesses}}$. This algorithm produces quite the same result of the normal scalar HD perceptron (70%) but it converges in fewer epochs.

3.5.5 Binarize SVM weights

As a first experiment, we have replicated the results of the paper [3] . We reached a 77.6 accuracy using a quite identical architecture. The front-end is a Support Vector Machine (SVM) with scalar activations and scalar weights. The weights are 10 hyperplanes with the same dimensionality of inputs, which separates the 10 classes. Hence the Matrix of weights has the same form of our Associative Memory. The aim of the experiment was to directly transform the weights (that produced the maximum SVM accuracy) into a binary AM. This turned to be impossible because of the distance between the query and all the hyperplane is minimal. To give an example, I kept randomly a scalar sample, and computing the L2 distance with respect to the AM I got: [113.2524, 113.2808, 113.2447, 113.2242, 113.0772, 113.2641, 113.1665, 113.2710, 113.3297, 113.3143]. The smallest one is 113.0772, then this is my guess and this is equal to the label, therefore the scalar architecture

would guess it correctly. The issue is that the second smallest one is very close to the correct one = 113.1665. 113.1665/113.0772 = 1.0008, so If I change this factor more than 0.0008 than the guess will be flipped and I would got a wrong result.
Given The Johnson-Lindnstrauss Lemma, the guassian Random Projection guarantees to maintain distances between any couple of points with a maximum distortion factor of $(1+\epsilon)$ if $m > 9 * \epsilon^{-2} * log(p)$, where p is the number of samples to be projected and m the final dimension. With the kind of features that we used for svm, we need ϵ very small and so m very large (in the order of millions). Indeed, in order to maintain the distances with probability = 1 we should force $\epsilon < 0.0008$ (1.0008-1). Given epsilon and p = 50000 then $m > \epsilon^{-2} * \log(p) = 7.000.000$; m should be higher than 7 million! Finally, the random projection maintain distances only if the distances are pretty different from each other. This explains why this experiment produced bad results (closed to random guess).

3.5.6 Correlation intra-classes

In all our previous experiment, we never take care during the training about the correlation between the data of different classes. For instance, if a feature of an image gives the same results among all the classes, then that feature has not meaningful information for classification, so we can associate to it a weight equal to zero. This is done at different levels in this paper [29] based on binary features obtained by random projection. Let's call n = num features, p = num samples, M = num layers Their input data X is in the space $\mathbb{R}^{n*p}$ instead the linear map A $\rightarrow \mathbb{R}^{m*n}$. So the projection results in Q = sign(A*X) $\rightarrow \mathbb{R}^{m*p}$.
At each layer "l" they choose "l" features between the m available ones so Q[index] -> $\mathbb{R}^{l*p}$. For each class, they denote the number of training points with equal unique sign pattern. For instance, at layer 2 they counts how many couples of [1 1],[1 -1],[-1 1],[-1 -1] are present in Q[index]. After that, they evaluate the membership index parameter "r(l,i,t,g)" that takes care of the relation between classes (where t is the number of unique sign pattern and g the number of classes). This is repeated m times per each layer, choosing randomly the indexes every iteration. For each layer, for each feature, for each unique sign pattern, for each class they learn a weight "r". So they store many scalar weights that they will use during testing. The query is projected and then all the correspondent weights are added

together to get the final score. The class that reach the highest score is the guessed one. For face recognition they use 150 layers and they get 95% of accuracy.
The HD translation is not straightforward, because they do not learn binary values but floating point weights, but the idea of correlating the feature of different classes can be exploited in HD using the Fisher score.

Fisher score

The Fisher score [30] takes into consideration the feature correlation between classes by selecting a subset of features which has more meaningful power in classification. It is somehow similar to the aforementioned paper, but also the weights associated to the HD features are binary (where 0 means untaken, and 1 taken). The fisher score is computed as:

$$F(X^j) = \frac{\sum_{k=1}^{c} n_k(\mu_k^j - \mu^j)^2}{(\sigma^j)^2}$$

where n_k and μ_k^j are the size and mean of k_{th} class, μ^j and σ^j are the mean and standard variation of the whole data set. It is also possible to compute the score in a one-vs-one mode, because of a feature could be discriminative for just a couple of classes and useless for all the others.

3.6 Analysis and results

The main pipelines and results explained in this chapter are shown in table 3.1. The best result is obtained of course with the completely scalar architecture (B) as proposed in [3]. The normalization, whitening and sliding steps are identical in all the pipelines. The last ones (E,F) use the Random Projection instead of the k-means clustering algorithm in order to produce the Dictionary of centroids (see sec. 3.5.1). The comparison between patches and Dictionary is computed by matrix multiplication, so the distance metric is the dot product. To make the results comparable, E and F use a larger dictionary because of they don't need a quantization step which increases the dimensionality by a factor 10. The only differences between (A) and (C) is the front-end. The first one maintains the linear SVM, hence it is clear that the quantization and binarization step causes a 16%

loss in accuracy with respect to (B). Including the binary HD perceptron instead of SVM (C), we get a further slight loss. So the SVM performs better than HD perceptron on this data set (in other cases, when the SVM suffers a lot of overfitting, the binary HD Perceptron is preferable). If we remove the final binarization of the activations (D) before the front-end, the HD perceptron increase of course its accuracy. As reported in the table, the Associative Memory is still binary; it is a mid-way trade-off between accuracy and computational effort. The distance computation between two HV is much cheaper if one of them is binary; hence it needs just a sign-xoring and an accumulation, without any scalar multiplication.
Looking at (D) and (F), it is understandable the Random Projection is a good alternative of K-means, if we need to speed up the training. Moreover, the training accuracy is close to the testing accuracy, so (F) is more generalizable and less prone to overfitting.

3.6.1 SVM Overfitting

The training accuracy was very close or equal to 100% in all the results that I obtained with the svm and 64000 features (sec. 3.5.1) . Just to remind: 1600 scalar centroid distances binarized into 10 bits with the thermometer code, and all the patches pooled over 4 quadrants in a final image HV $\rightarrow$ 1600*10*4 = 64000 bits. The issue is that also changing the parameter C for the regularization, the overfitting does not change. This behavior is explained by analyzing the svm algorithm in a bipolar HD space. First of all, a so huge number of feature is always prone to overfitting, it is highly probable that exists an hyperplane able to divide one class from the others, but hardly ever this separation is generalizable because of it is usually based on a small subset of features which is suitable only for the training set. In the Real space the C parameter defines the hardness of the margin; a small C means soft-margin, so the SVM can admit misclassification in favour of larger margins. In the binary space the regularization parameter does not have effect. Indeed, each feature can be represented in 2-D as square centered in the origin, where the possible values are just the corners ($x, y = \pm 1$). It is clear that the hardness of the margin is quite meaningless, because there are not outliers which can be misclassified: all the points placed in the corner are taken or not, there are not mid-way tunable with the C parameter.

	BACK-END				FEATURE EXTRACTOR			
A	normalization R^32*32*3	whitening	sliding R^6*6*3	kmeans	patches*D R^1600	norm.	quantization N^1600	binarization D^16000
B	normalization R^32*32*3	whitening	sliding R^6*6*3	kmeans	patches*D R^1600	norm.		
C	normalization R^32*32*3	whitening	sliding R^6*6*3	kmeans	patches*D R^1600	norm.	quantization N^1600	binarization D^16000
D	normalization R^32*32*3	whitening	sliding R^6*6*3	kmeans	patches*D R^1600	norm.	quantization N^1600	binarization D^16000
E	normalization R^32*32*3	whitening	sliding R^6*6*3	RAND proj	patches*D R^16000			sign D^16000
F	normalization R^32*32*3	whitening	sliding R^6*6*3	RAND proj	patches*D R^16000			sign D^16000

ENCODING			FRONT-END		TRAIN ACC	TEST ACC
Pos. binding	pooling 4 N^64000	binarization D^64000	linear SVM		100	61.5
	pooling 4 R^6400	norm.	linear SVM		86.1	77.6
Pos. binding	pooling 4 N^64000	binarization D^64000	HD	PERCEPTRON	81	58.8
Pos. binding	pooling 4 N^64000		HD scalar	PERCEPTRON AM binary	83	64.5
Pos. binding	pooling 4 N^64000	binarization D^64000	linear SVM		100	52.8
Pos. binding	pooling 4 N^64000		HD scalar	PERCEPTRON AM binary	66	63

Table 3.1. Pipelines and results. Moving towards the step of the pipeline, when the representation space changes, it is reported in the format $\mathbb{X}^{m*N}$. $\mathbb{X} = \mathbb{R}$ is the real space, $\mathbb{N}$ the natural space and $\mathbb{D}$ the binary space (0,1 or -1,+1). The main differences to be noted are: kmeans + quantization substitude by RP in (E) and (F), binary (A,C,E) or scalar (B,D,F) encoding, SVM (A,B,E) or HD (C,D,F) training.

3.6.2 3D-space representation

Representing the HD space in three dimension is a good tool for the analysis of the architecture. First of all, it makes it simpler to recognize the complexity of the classification problem and the goodness of the feature extractor. In the fig. 3.5 is shown a subset of CIFAR10 samples projected from space $\mathbb{R}^c \rightarrow \mathbb{R}^3$ (where c = number of centroids or elements of the Dictionary). We used the TSNE python function (package manifold of sklearn library) to convert similarities from high-dimensional data into 3D. The plots underlines the effect of the normalization step after the Dictionary comparison. As already said in the previous sections, the vector generated after the Dictionary is composed by c features; each feature represents a distance between the patch and the correspondent centroid. Without re-scaling all these distances, the accuracy is strongly affected. This is also clear from the figure where each class is associated to a different colour: the upper plot has all the classes merged in the 3D space, making impossible the classification problem (especially using a linear algorithm), the lower plot instead shows that with the scaling step all classes are better separable. Moreover, it is possible to identify more than one cluster for each class, in particular the orange one; this would suggest to use two different cluster of that class, otherwise the HD will suffer surely the clustering issues, shown in fig. 2.8, indeed the mean of all orange samples (HD vector) is far away from each each orange cluster.

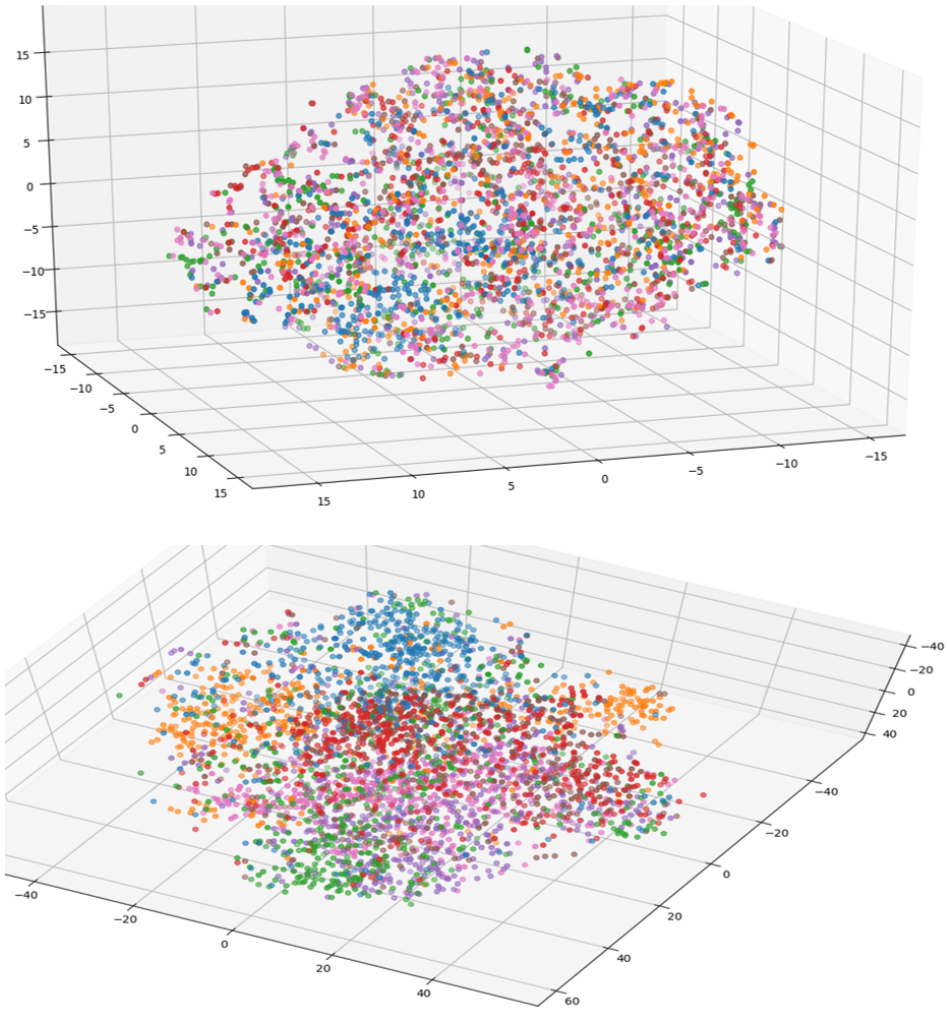

Figure 3.5. The effect of scaling the features after the Dictionary comparison. The upper plot (without scaling) has all the classes merged together, making the classification task really difficult. The lower plot (with rotated axis for a better cluster representation) shows instead all the ten classes of CIFAR10 (with different colors) distributed in clusters into the reduced HD space. It is possible to recognize two clusters of class "orange" and some critical zones of overlapping (pink and violet classes).

Chapter 4

BNN + HD

In the previous chapter, we have slightly increased the complexity for a huge gain in accuracy with respect to the first architecture. Nevertheless, we are still far from the state of the art ruled by the Convolutional Neural Networks (CNNs). To perform a further step towards better accuracies, in this chapter we will take into consideration the Binarized Convolutional Neural Network (BNN) as feature extractor.

4.1 Background

There are different works focused on the binarization of CNNs, but first we will introduce how a typical CNN works, and afterwards we will describe one possible method to binarize both the activations and the weights of the network. The following sections takes a cue from the following sites where it is possible to find more detailed information [31–34].

4.1.1 CNN

The Convolutional Neural Network is a deep artificial neural network, with a feed-forward propagation (no feedback interconnection during the inference). It is inspired by biological processes. Indeed the neurons connectivity simulates the organization of the animal visual cortex making this kind of architecture optimal

for Computer Vision tasks.
Each neuron is fired by stimuli belonging to a restricted zone called receptive field. Anything outside the field cannot affect the status of the neuron. A receptive field of a neuron usually overlaps the field of adjacent neurons to maximize the coverage of the visual space. This concept is mathematically represented by a filter which is moved and applied over all the input image. The size of the filter defines the boundary of the receptive fields, and together with the shifting step (stride) of the filter, they defined the degree of overlapping. Each filter has three dimension: width, height and depth. The depth of the input RGB image is equal to the number of channels (red, green, blue). When the filter is applied on the receptive field, then it will produce just one value which will activate the correspondent neuron on the feature map.
The fig. 4.1 shows an example for clarification: the receptive field is the coloured part of the input image and it has the same dimension of the filter. The filter can be applied in four possible positions (the corners) if a unitary stride is used, providing a single value in the feature map for each of those positions. Mathematically, the application of the filter is the component wise multiplication followed by the sum of all the components. A filter will produce a single Feature Map that

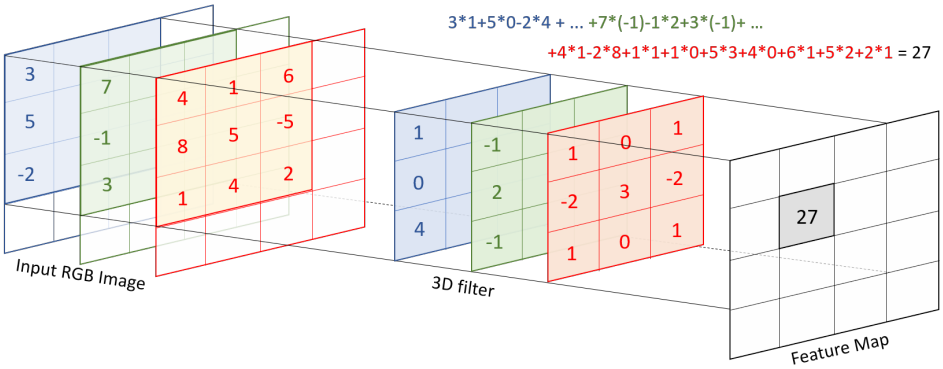

Figure 4.1. In the first convolutional layer, a filter is applied to the input RGB image. The dimensionality of the filters depends on the size of the receptive field (3x3) and the number of channels of the image (3). When the filter is applied, it will generate only one value which will be store in the Feature Map. The mathematical component-wise operation is reported in the upper-right corner.

will represents our new image from which another CNN layer could start. Usually many filters are applied for each convolutional layer, in such a way the new image will be composed by many channels (one for each Feature Map); typical values are in the order of hundreds. This fact allows to arbitrary change the number of channel moving from one layer to another.

Padding

Following the aforementioned method, the generated feature map will be smaller than the input image because we cannot apply the filter on the image boundaries. Sometimes this is useful and allows not to use a pooling layer; if we want to maintain, instead, the same size of the image, the padding is inserted. The padding is a zero-outline around the image, in such a way it will have rows and columns on the boundaries all filled with zero values (fig. 4.2). The size of the padding is arbitrary and usually depends on the size of the receptive fields: to maintain the input dimensionality the padding's width should be equal to $\frac{\text{width of filter}-1}{2}$.

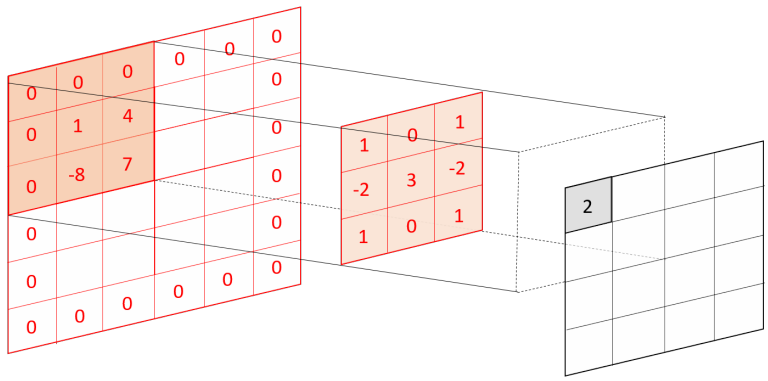

Figure 4.2. An outline of zeros and a 3x3 filter will produce an Feature Map with a width and height equal to the input image.

Pooling Layer

The pooling is a technique commonly used to reduce the size of each feature map. It moves a window over the whole feature map just like the filter, but unlike the Convolutional layers, the windows never overlap. Each window is collapsed to a

single value by computing the average of all the components, or keeping just the maximimum value. The number of channels remains the same.

Typical Architecture

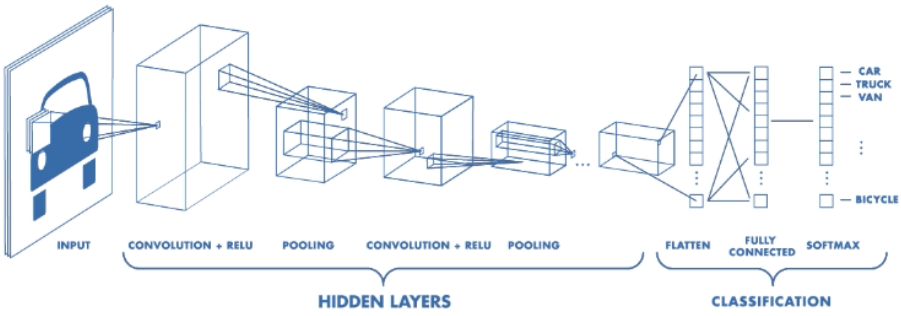

Figure 4.3. Typical CNN architectures: convolution, ReLU and pooling to extract meaningful features; FC and softmax to compute the final classification. Note that the convolutions are represented as parallelograms because the filters are 3D and covers all the channels; the 2D pooling instead is applied on each Feature Map separately. Image source [35].

A typical CNN architecture for image recognition is really deep. As shown in fig. 4.3, it usually implies many layers of different kind: convolutional, pooling and fully connected. The architecture is generally divided into two parts:

1. A sequence of Convolutional+ReLU and Pooling layer aim to extract meaningful features from the image. The first layers capture low-level local patterns such as edges and textures; we could say that they applied different preprocessing techniques on the same image. The higher hidden layers instead learn category-level invariances features which are particularly useful for the specific classification but less generalizable to other data sets [36].

2. One or multiple Fully Connected (FC) layers which will finally combine the features for generating high-category-level concepts which will produce the final classification.

A final softmax layer will transform the output of the last FC layer in a normalized vector of probabilities, which assigns a confidence level for each class. The class with highest confidence would be the guess. The softmax function is the following:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Usually, the number of channels is progressively increased, concurrently the size of the image is decreased by applying pooling layers on the feature maps. Before the final classification step, all the feature maps of the last hidden layer are flattened in a very long one dimensional vector (something similar to a scalar HV). Hence, the final number of channels and the size of the feature map should be carefully designed, in order to avoid a dimensionality explosion.

ReLU non linearity

The Rectified Linear Unit is a non-linear function which is usually applied after each convolutional layer. The reason is that the convolution is just a linear operator composed by multiplications and addictions. The representation of the real world is often non linear, hence a non-linearity have to be inserted. The function is really simple: all negatives values are flattened to zero, all the positive ones are left untouched.

Training using Backpropagation

The training of a CNN follows the same rules of whatever neural network. The filters of each convolutional layer are represented by 3D weight matrix which are randomly initialized. This filters will be learned by backprogating the errors produced by the softmax function during the forward propagation (identical to the inference/testing). The ideal outcome is a vector of all zeros and a single component equal to one which corresponds to the correct class. In the real case, we will obtain a vector of probability (degree of confidence) which defines how much the weights of the last layer need to be modified. The total output error is computed as sum of the square distance. This error is propagated back to the inputs. All the gradients of the error with respect to the weights need to be computed, and then all the convolutional filters and the weights of FC layers are updated by means of

the gradient descent (fig. 4.4). The gradient computation is performed for each image; hence it is usually the bottleneck of the training process. The update takes place once every patch, so the patch size indicates the number of images which are tested using the same weights.

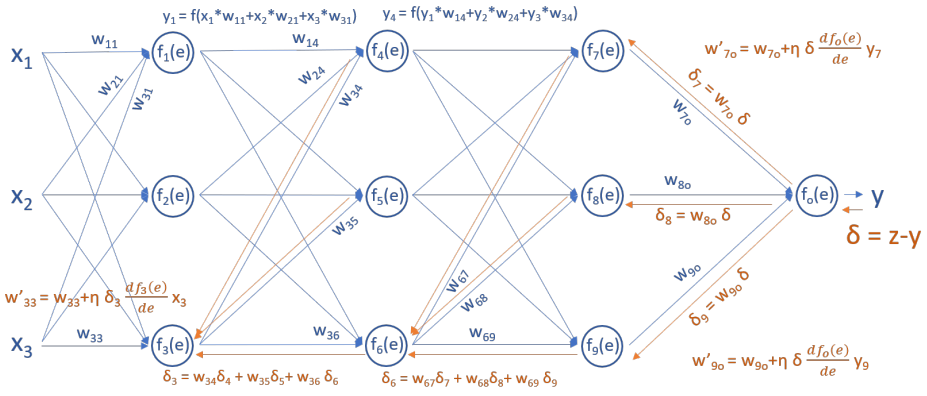

Figure 4.4. The NN training is divided into three steps: (1) forward propagation, blue arrows (2) delta error backpropogation, red arrows partially shown (3) weights update, gradient descent formulas. z is the target and η is the learning rate. Only some equation are shown but y_m and δ_m are computed for each node, and every weights are updated of course.

4.1.2 BNN

On the one hand, CNNs can achieve really good performance in image recognition, even when applied to a large-scale datasets such as IMAGENET (1000 categories and 1.2 million images [37]). On the other hand, the main drawback is the computational cost, which requires a huge amount of memory and floating-point operations. To make an example the AlexNet [38], the winner of ILSVRC 2012 competition, had a large impact on computer vision proposing a really deep CNN designed to be essentially run on GPUs because of its 61 million weights and 1500 million operations. In the last years, many works focused on this issue and many architectures was suggested, to make the CNNs less computational intensive. The most promising works were carried out by Courbariaux et Al., which found a method to binarize the weights [39] and later also the activations [40] with minimal losses in

accuracies on small datasets. Unfortunately, this Binirized Neural Network (BNN) does not scale as far as to large datasets. Other works relaxed the binary constraint to increase the accuracy on IMAGENET: the Dorefa-net [41] uses ternary representation of weights, or more generally it quantizes the weights and the activation to an arbitrary number of bits. The Xnor-net [2] instead associated a single scalar factor to each binary filter. We used this last neural network as our baseline.

4.1.3 Xnor-net

The Xnor-net presented a binarization technique strengthened by a scalar factor to minimize the quantization error. They approximated each weight filter as $W \approx \alpha B$, where $\alpha \in \mathbb{R}^+$ and B is a binary filter with the same dimensionality of W $\mathbb{H}^{\text{channels}*w*h}$. To be noted that W is not the matrix of weights; all the weight filters W of layer l will compose the matrix and for each of them α is different. However, the scaling factor has a minimal influence on the computational cost, because it is applied after the binary matrix multiplication: $I * W \approx (I \oplus B)\alpha$.
By minimizing the cost function $J(B, \alpha) = \|W - \alpha B\|^2$ they found that the optimal solution is $B = \text{sign}(W)$ and $\alpha = \frac{1}{n}\|W\|_{l1}$ where $n = \text{channels} * w * h$ which is the length of the HV obtained by flatten the filter weight.
The training is divided into four main steps.

1. For each filter: computation of α parameter, B and $W_{approx} = \widetilde{W} = \alpha B$
2. Forward Propagation: Each convolution is done with binary weights because the activations are binarized just before each layer. In the paper they compute also β factors from the binary activations and the multiply them to the α factors: in other words they substituted n=(channels*w*h) multiplications between the filter and the receptive field with just one multiplication $\alpha * \beta$. In the Pytorch implementation of Xnor-net that we found online [42] and we used as a baseline, the β factors are never computed and so the inference is totally binary. The *alpha* factors instead are used only during backward propagation (fig. 4.5).
3. Backward Propagation: $\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial \widetilde{W}_i}(\frac{1}{n} + \frac{\partial \text{sign}}{\partial W_i}\alpha)$; considering a blunter sign step, the derivative of the sign function saturates all values over/under ± 1 and maintain constants all the others.

4. Update Parameters: using high-precision scalar weights W otherwise the gradient descent would ignore the computed gradients which are values usually too samll.

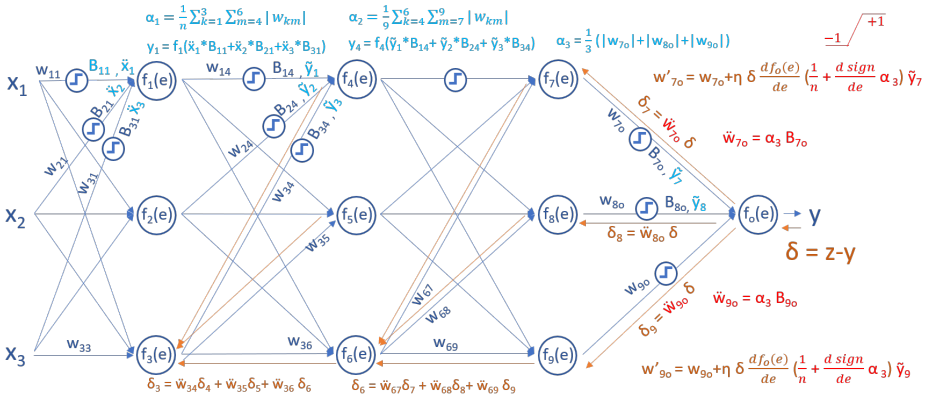

Figure 4.5. By comparison to fig. 4.4, during the inference and the forward propagation all the weights and all the activation are binarized before each layer. α factors and approximated weights are used during backward propagation. Fully precision weights are updated, modifying the sign function for computing the gradients.

First and Last scalar layers

In the paper is specified that the first and last layer are not binarized because the speed up would not be considerably high, due to the small channel size (3) of first layer and the filter size (1x1) of the last one.
However we need to take into consideration that the software implementations of BNNs have a limited performance gain. On GPUs the speed-up is limited to around 5x. It depends a lot on the instruction set and the number of cycles relevant operations take. E.g. ARM Cortex M series uC do not have a popcount instruction, so it takes 7 cycles to compute the 32-bit popcount and it thus limits the speed-up. In SW, it used to vectorize in the direction of the channels, i.e. pack 32 different channel results into one 32-bit word at any pixel of the feature map. This makes processing of the first layer inconvenient, so it needs a separate implementation

and yet will not achieve as high gains as in the other layers of the network.
If we consider specialized HW, it is very reasonable to add a BNN accelerator, bringing tremendous speed-up and energy efficiency gains. Running non-BNN operations at this point is orders of magnitude costlier. Just to have an idea, let's compute a rough estimation. In [42] the first layer has 32x32 RGB input, kernel size = 5, padding = 2, output channels = 192. This means that it needs 5x5x3 filters which will be applied 32*32*192 = 196608 times (the feature map has the same dimensionality 32x32 of the input because of the padding). This implies a huge amount of operations, which can surely be reduced by decreasing the kernel size and removing the padding. Nevertheless, this number of operations is comparable and sometimes also higher than the other layers. However, to binarize the first layer cause a huge drop in accuracy.

Recently it was experimentally demonstrated that the dot-product in the first layer are not well preserved after the binarization "due to the non-random orientation of the input data relative to the axes of binarization" [43]. For this reason a different approach is necessary, but it has not been proposed yet. A possible idea based on HD is presented in the future work section.

The last layer, instead, is effectively less computationally intensive due to the kernel size but also to the small number of output channels: 8*8*10=640 1x1x192 filters. Nevertheless, the xor computation is drastically cheaper than the scalar multiplications, the last layer alone computes more than 25% of operations with respect to the total ones (except the first layer), even if the memory saving is just 2%. So, even if it is absolutely preferable to binarize the first layer in order to maximize the gain, we left this task to the community. We focused instead on the last layer, which is interesting for our experiments for two reasons:

1. Having an HD front-end is useful for Knowledge Representation purposes.

2. It is possible to substitute the last layer with the HD and then moving backward, trying to remove layers as much as possible, evaluating how good the image representation is in the hidden layers.

4.1.4 Nin-net

The Network used by [42] is the Network in Network also called NiN-net [44]. It has this name because of its particular architecture which integrates multilayer perceptrons inside a convolutional layer producing the so called Mlpconv layer. The convolution remains the same, a filter is applied to a receptive field and it generates a value on the feature map. The difference resides on the filter which is replaced by a micro network with two hidden layers (fig. 4.6). It was demonstrated that a multilayer perceptron with two hidden layers, enough number of nodes and non-linear activation functions can ideally approximate whatever function [45]; starting from this consideration they apply a small multilayer perceptron to each filter in order to compute more abstract and complex features from local patches. In this way, the task for higher layers becomes easier, because their complex features inputs are less than lower-level feature which is usually redundant and overlapping, so all the possible combinations in a smaller set are drastically lower.

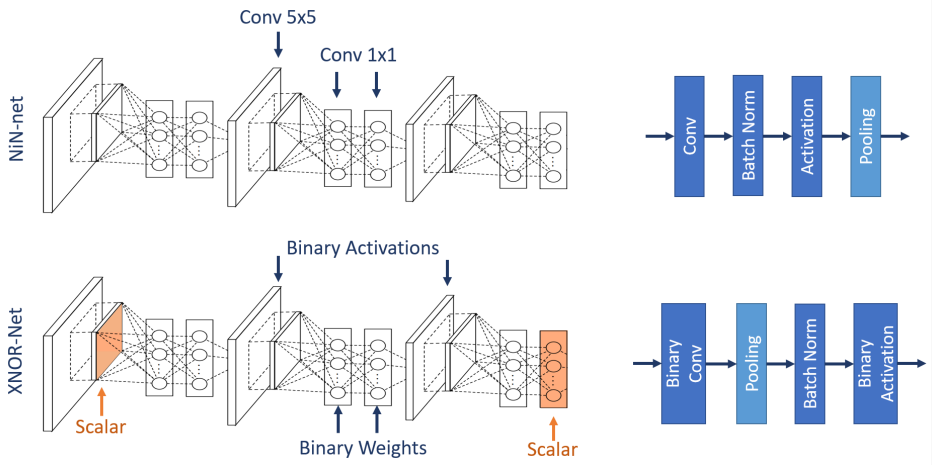

Figure 4.6. In the upper half, the representation of a Mlpconv layer: a convolutional layer composed by complex filters based on micro Network. Source [44]. In the lower half, the main modification applied to the NiN net in order to binarize both the activation and the weights.

1x1 Convolutions substitute Multilayer Perceptron

The multiplayer perceptron inside a filter is easily implemented by pipelining two convolutional layers with kernel 1 to a first convolution with an higher kernel. Indeed, let's consider a perceptron layer: each node is represented by a linear function $f(x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4)$. The same linear function is used for filter application; after the component wise multiplication $x_i * w_j$, all the terms are added together to generate a single feature. Therefore, each node of a fully connected layer is practically a filter. If the kernel size of a convolutional layer is one, then the filters are applied only on the channel dimension which is composed by all the filters of the previous layer applied to the same feature map (fig. 4.7). The result is that they are a linear combination of all the nodes of the previous layer. This is exactly the structure of a multi-layer perceptron.

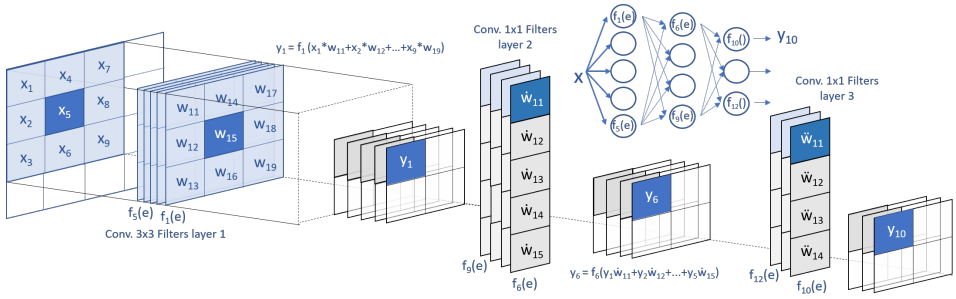

Figure 4.7. The first layer has a 3x3 kernel and applies a set of 5 filters on all the receptive fields (this number does not refer to the real structure). Each receptive field is represented by a row vector in the channel dimension. A second conv. layer with kernel 1x1 applies its filters only on the channel dimension, so it combines linearly all the values belonging to the same receptive fields. The same for the last convolutional layer. In other words, a complex filter for each receptive field is computed.

NiN Architecture

The NiN-net is divided in three Mlpconv blocks (fig. 4.6) linked by pooling layers. Each block is composed by three convolutional layers, but only the first one has a non-unitary kernel. The convolutional layers are interleaved with relu activation,

batch normalization layer and binarization of activations. The normalization would produce a zero-mean vector which reduces the binarization error. Actually, the pooling layer is not placed between one of the aforementioned layers, but it is moved slightly backward before the batch normalization step; otherwise applying a max pooling on a binary matrix would result in a matrix with most of its elements equal to 1. Input and output channels are arbitrarily chosen, we maintain the same structure of [42] which never overcomes the 192 channels per layer. The padding is always chosen not to modify the feature map size, which is reduced instead by two max pooling layers by a x2 factor and by a final average pooling by a x8 factor. In this way the dimensionality of the output layer depends only on the number of channels.

4.1.5 Transfer Learning

Training a CNN requires a huge amount of time, especially for large datasets. Moreover, it has some parameters to be set which are quite never specified; such as how the learning rate is adjusted during epochs and the weight decay of the optimizer. Hence, starting from a pretrained baseline is strongly recommended to speed up the training and to produce comparable results, if the aim is to slightly change the architecture. However, the CNN can do something more. If the CNN is trained with a quite large and various dataset, the first convolutional layers are able to extract meaningful low-level features whatever is the input data, so they are fully generalizable. It is possible to reuse these layers as a global feature extractor and train the higher layers for the classification specifically for the target dataset: this is called transfer learning. In [46] they even transfer knowledge from one device to another with different type of inputs: video, audio and inertial. They force all devices to reach the same Latent feature representation at some point of the network. The Latent feature representation is given by a trained network which is therefore able to transfer its knowledge to other devices without the necessity of new labelled data. The higher level are instead task specific and they are trained separately with a small number of labelled data.

4.2 Experiments

Starting from the consideration that a more powerful feature extractor is necessary to improve the accuracy of the Kmeans-HD architecture, we moved to those binary algorithms which are the closest to the state of the art: the Binarized Convolutional Neural Networks (BNN). Our feature extractor cannot be the BNN as it is, because we would not have any gain in complexity; so the aim is to cut the BNN up to some internal hidden layer and check if that kind of features are suitable for our HD architecture. However, for the aforementioned discussion, just removing the last layer would result in a 25% gain in complexity, because of its scalar weights.

4.2.1 Training from scratch

As first experiment, I substituted the last scalar convolutional layer with a binary associative memory. Then I trained the HD and BNN weights, randomly initialized, in two different ways: separately or together.
The input of the last scalar part of the network has dimensionality 192x8x8 and it is composed by: Batch Normalization (BN), scalar Conv2d layer (192,10) $\rightarrow \mathbb{R}^{10*8*8}$, Relu, AvgPooling (kernel = 8) $\rightarrow \mathbb{R}^{10*1*1}$. Our first approach for binarization of the last layer was the following:

1. The BNN is cut just before the last Conv2d which uses scalar inputs and scalar weights. As first experiment, I left the final AvgPooling layer which produces a 192x1x1 output vector.

2. The HD layer is added, it consists of: Batch-norm + RP + Binarization + AM distances. The $\mathbb{R}^{192}$ vector is projected in the HD space ($\mathbb{H}^D$, D = 1000 or 10000) by a Random Projection and it is compared to the HD Associative Memory.

3. The RP is fixed during all the training.

4. The BNN and the HD are alternatively trained: the BNN with gradient descent AM fixed, the HD instead with the one-shot HD learning and fixed weights of the network.

5. To generate the final output, I computed the distances multiplying the query in HD space by the AM. They are both binary so their norms are constant, and so the matrix multiplication query x class is proportional to the cosine similarity. Then I applied the softmax on the 10 distances. The final loss is computed on this final vector using the CrossEntropyLoss(), as in the original code.

Learning this architecture from scratch was very difficult. This kind of learning is slow, it starts with a low accuracy (15%) and it grows slowly converging to 40% after the 60th epoch. The training accuracy remains just slightly higher than the testing one. Therefore, it seems that, if we include the RP and AM, the BNN is not able to train the weights in the right way. If I train also the HD with gradient descent, the result is quite the same, even a bit worse. I could learn the RP instead of generate them randomly but at this point, we have to store also all these RP weights. The architecture would have to gain neither in memory nor in number of computations. In the end, this was not a winning approach, the maximum accuracy that I reached is 50%.

4.2.2 Training HD on pretrained BNN

A good alternative was to move the BNN cut further back into the net, overcoming the pooling layer. Hence, removed Conv2D, Relu, and AvgPooling. I binarized the 192x8x8 output after the BN and I used it directly as HV. Indeed, the RP is removed and a direct binarization of the flatten activations will produce an HV with a proper HD length (D = 192x8x8 = 12288 bit).
We used the pretrained weights (Transfer learning sec. 4.1.5) of the original BNN which gives 85% in accuracy. Using these learned weights, I trained the HD Associative Memory ($\mathbb{H}^{10x12288}$). The one-shot training gives a result of 66% in training accuracy and 65% in test accuracy. After The one-shot training I applied the HD perceptron, which was able to reach the 80% in test accuracy, so just a 5% of loss using the scalar Conv layer.

4.2.3 Loss function

To further improve the accuracy of the BNN+HD architecture, I slightly modified the loss function of BNN to force producing HV with the same density of -1s and 1s. The BNN training is slightly improved by this factor, but it is not necessary: to produce HVs of equal density is sufficient to multiply the HVs generated by the network for a random HV, which is maintained fixed for all the queries.
Analyzing the output of some internal layers, I found that some features remained constant among all classes, so I added an additional loss term, which allows to obtain more discriminant bits. This additional term is computed as: (number of constant features in a patch)^2. In this way, the HD perceptron works better and it was able to reach 94.5% in training accuracy and 83.6% in test accuracy.

4.2.4 Transfer learning on Cifar100

Another experiment concerned the generalization capability analysis of our system. The HD is in fact really suitable to be adapted to any change in the number of classes. It is possible to dynamically add or remove classes from the AM and retrain them in a very fast one-shot way. Unfortunately, the final result strongly depends on the Cifar10 network. We have tested for instance the Cifar100 (an extended CIFAR10 dataset composed by 100 classes instead of 10 [47]) using the same network trained with Cifar10. We got 40% in accuracy that is not too low, considering that the BNN training from scratch for this dataset reached 60% and that some classes of Cifar100 are totally different from the trained one (as that one related to food). It would give better results to train the HD part using as BNN a network trained for IMAGNET dataset, which is much more generalizable for its number of classes and images.

4.3 Analysis and results

In this section, I am going to describe and comment the main results of our experiments. During the discussion will be also given some clues for useful application of HD to BNN.

4.3.1 PReLU and tricks

During my tests I found a paper full of inputs for optimizing BNNs [48]. They proposed a set of modifications to be applied to any BNN in order to binarize also the last scalar layer with a declared loss of 1.3% with respect to DoReFa-net for ImageNet. The main point is the suggestion of substitute all ReLU activations with PReLU ones and to add a Batch Normalization as final scaling layer right before the softmax. Including also my loss term for discriminant bits I obtained very good results, practically without losses or also better: I replaced the last layer with a conv binary layer and I got 86.32%.

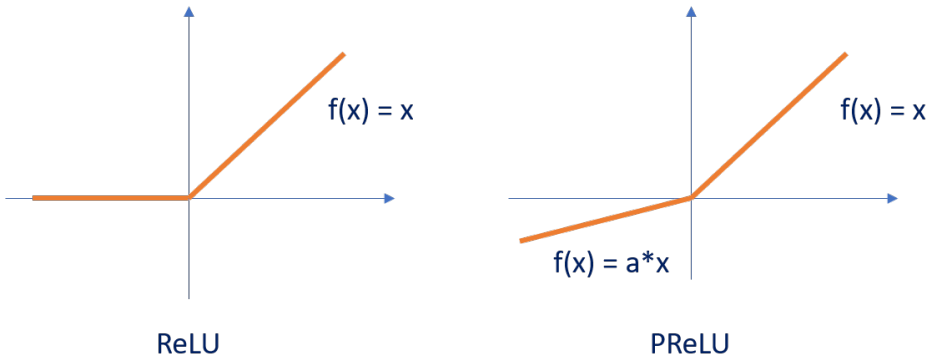

Figure 4.8. Differences between PReLU and ReLU activation functions.

What does the PReLU do? It is a Partial ReLU (fig. 4.8), essentially a ReLU but the negative values, instead of being set to 0, are multiplied by a small coefficient ($<$ 1). The PReLU is an activation function, so it has to be computed during inference, at run-time. Nevertheless, we can get rid of this multiplication by modifying the binarization threshold. This is valid also for the normalization scaling factor that is learned during the training. Let's say that A is a feature generated by the convolutional layer, ρ is the coefficient of PReLU and γ, β are the scaling factor of the BatchNormalization; then the binarization should define if $A * \rho * \gamma + \beta > 0$ or not, but we can avoid all multiplications by performing A$> -\frac{\beta}{\rho * \gamma}$. Of course, this would require a tunable comparator with enough precision, but ideally, it does not need additional computations during inference.

Therefore, the binarization of the last layer can be effectively obtained with this trick and the gradient descent. Nevertheless, the HD has other advantages, such as the dynamical allocation of classes and the fast one-shot training which does not need any re-computation of the BNN and any scalar factors. In fact neither the normalization nor the ρ factor are used during inference, leaving the architecture complete binary and without any comparator that would be surely an issue during design.

4.3.2 More than one layer

I have analyzed the effect of moving the BNN cut at different layers. I have found that by removing the last-two layers I maintain the same accuracy than removing a single layer, even better (83.7). Going further back, the accuracy gradually decreases (fig 4.9). If we insert the PReLU activation function the one-shot learning is improved, mainly in the first layers. Nevertheless, the Perceptron reached lower accuracy at higher layers. This fact underlines the high dependence of the HD with respect to the activations: two different networks, that performs equally in output, generates completely different features in the hidden layer. The results are reported in table 4.1.

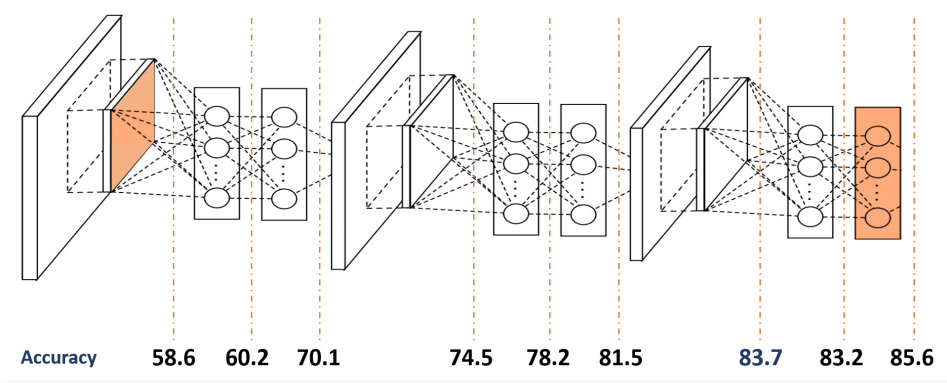

Figure 4.9. The network is cut and the activations are flattened. In the end, the obtained HV is trained by the HD Perceptron. This is a fast method (withour retraining) for analyzing the hidden layers and for binarizing the last layer.

It is noteworthy that it is also possible to cut the NiN-net inside the MlpConv block, without any drastic drawbacks. An exception is the first block, where there is a considerable loss of 15% before the second perceptron layer of the block, which is what I expected from separating a filter in two pieces. However, this is reasonable because at higher levels there is less necessity of complex filters, probably two MlpConv blocks would be enough for this dataset. This is also one of the possible usage of the HD, it could be a useful tool in BNN analysis to speedily verify the effectiveness of each layer.

# Layer	1	2	3	4	5	6	7	8
Dimensionality (k)	196	163	24	49	49	12	12	12
Kernel size	5	1	1	5	1	1	3	1
Accuracy one-shot	29.5	36.4	49.8	55.8	64.8	71.1	71.1	68.8
Accuracy perceptron	58.6	60.2	70.1	74.5	78.2	81.5	83.7	83.2
Acc. PReLU one-shot	41.9	46.2	57.0	56.9	68.0	72.5	73.4	72.0
Acc. PReLU perceptron	67.6	63.52	73.8	70.9	74.1	76.7	76.8	76.1

Table 4.1. The table shows how the accuracy changes when the HD is applied at different points of the CNN. It is particularly interesting the fact that the last layer has no additional information with respect to the previous one. The "dimensionality" means the number of bits composing each HV, obtained flattening the output of the convolutional layer. To be noted that the one-shot training is really high, mainly in higher levels. Adding the PReLU improves a lot the one-shot training, but the perceptron performs worse in the last layers.

4.3.3 Naive complexity estimation

The aim of this project is to find simple architecture for image recognition in order to consume power low enough for embedding. At the same time, this kind of algorithm should not lose too much precision in classification. Unfortunately, we did not find any architecture really promising and therefore we never moved toward HW implementation. For this reason, we can just roughly estimate the power consumption of the proposed architecture. The number of operation needed for each algorithm are easy to compute. Nevertheless, the scalar architectures, such as the first layer of the BNN, use MAC (Multiply and Accumulate) operations instead the HD and the binary layer of the BNN uses xor-popcount operations.
In order to make them comparable, we can consider the number of cycles required

by a microprocessor (with MAC and popcount instruction in the instruction set). It is easy to demonstrate that 32 MAC operations requires 32 cycles, instead 32 xor-popcount requires just one cycle. Therefore we can state that the xor-popcount instruction as a 32x gain with respect to the MAC operations. Nevertheless, if we suppose to design some dedicated HW the gain is much higher. At least from a power point of view, we can compute the number of gates necessary for each operation, which is somehow proportional to the power consumption. Of course, we should take into account many other factors which affect the power, but our assumption is sufficient for a rough estimation. For what concerns the popcount operator we can suppose to use HW dedicated in CMOS technology which requires about 10000 Full Adders to count the number of ones in a 10000-bit HV; each FA is approximately composed of 4 xor gates (2 xor, 2 AND, 1 OR gate actually but the xor requires more transistor in CMOS). Therefore computing the distance between two 10000-bit HV requires approximately 50000 xor gates.
For MAC operation we suppose to have 32 bit precision, the Area depends mainly on which multiplier and adder we want to use and usually it will increase when the propagation delay is reduced. Assuming to choose fast units, the complexity (number of full adder) would be approximately O(n^2) which requires approximately 1000 FAs, hence 4000 xor gates [49]. With the aforementioned consideration, which are just rough estimations, the Hamming distance computation between two 10000-bit HVs is equivalent to 12.5 MAC; in the same way 1 MAC operation corresponds to 800 xor-popcount, hence 800x power gain.
I computed layer by layer both the memory and computational requirements of the xnor-net based on NiN architecture. Starting from the last layer, I removed one layer at a time evaluating the complexity and the accuracy, then I compare them to the previous architectures (fig. 4.11). It is clear that the first layer produced a strong bias for the complexity of the BNN, therefore, even if we remove all the layers but the first one, the overall gain would be 30%. It is noteworthy instead that the kmeans+HD architecture has an higher accuracy with respect to the first two layers of the BNN and it has a complexity which is almost half with respect to the BNN. Based on the consideration that in the kmeans+HD architecture the patch slicing and the centroid Dictionary essentially implement a convolutional layer (fig. 4.10), we can state that the unsupervised training of those filters alone

works better than gradient descent.
To give an example of how the complexity is computed, let's consider the first BNN layer which has a kernel size = k = 5 and padding = 2, 3 input channels and 192 output channels.

1. Each filter has to compute: k * k * CH_{in} multiplications.
2. The number of filters per conv layer is the number of output channels.
3. Each filter is applied to the images w*h times (given k and padding).

So the number of $MAC = k^2 * CH_{in} * CH_{out} * w * h = 25 * 3 * 192 * 32 * 32 = 14.7 * 10^6$ MAC, which corresponds to $471 * 10^6$ xor (or $11.8 * 10^9$ xor with dedicated HW).

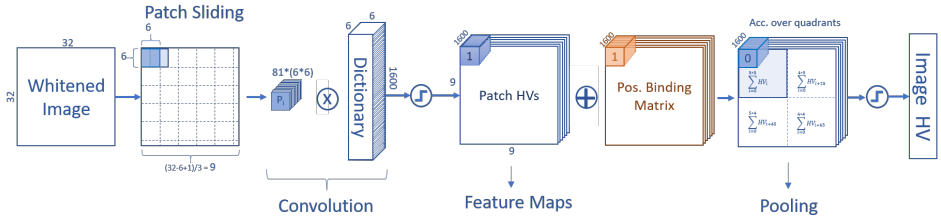

Figure 4.10. The Kmeans architecture is essentially a convolutional layer. The dictionary represents the weights of the layer where each centroid is a filter, the patch sliding allows the convolution. The accumulation over quadrants is a pooling layer.

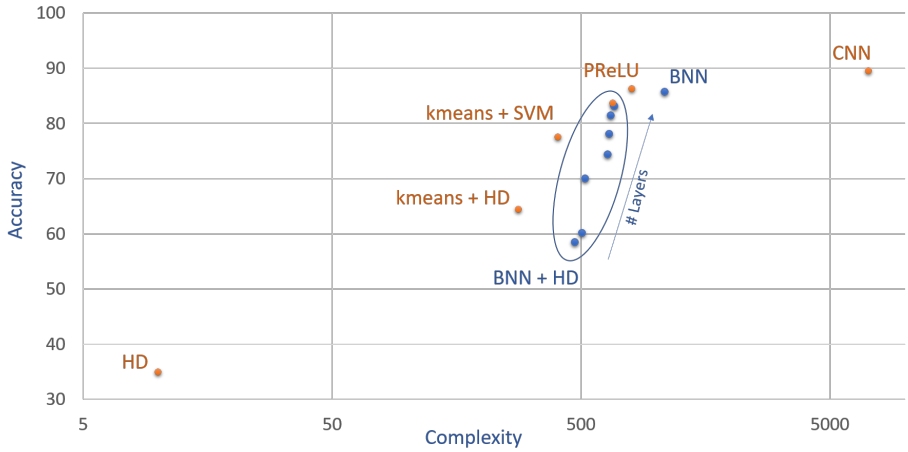

Figure 4.11. Accuracy-complexity trade off.

Chapter 5

Activity recognition

Egocentric Activity recognition is another task in the Computer Vision domain. It aims to recognize actions and movements of one or more subjects during a normal real life setting. Since the recognition of an activity is essentially based on the recognition of particular sequences of video frames, which is nothing but an image, we had already all of the necessary means: BNN for the image encoding, HD for the sequence encoding. As a baseline we used this paper [50] which uses a pretrained CNN as feature extractor and the Echo State Networks (ESNs) for encoding the temporal information of two different datasets. Our goal is to use a BNN instead of the CNN and the HD permutations to encode the sequence of frames.

5.1 Echo State Networks

The Echo State Networks (ESNs [51]), also known as Reservoir Computing (RC), is a novel Recurrent Neural Network composed by three layers: input, output and a hidden reservoir layer with several random connected internal units. Input-hidden and hidden-hidden connections are random and fixed. Only the hidden-output connections are trained. This speeds up a lot the training process, which is faster than other RNNs but it requires a higher amount of hidden weights. The main important thing is to set carefully the connections, in such a way that they do not explode or die. Therefore, it is necessary to set the spectral radius to one. Another

important factor is the sparseness of the hidden layer. In practice, the ESN is represented by three matrices W^{in}, W^{out} and W^{x}. Each frame would generate $x = in * W^{in} + x(n-1) * W^{x}$ and $y = W^{out} \sum x$.

5.2 Datasets

In this section I am going to describe the two datasets we used, the same proposed in [50]: DogCentric [52] and UECPark [53]. The former is composed by 208 videos of four different dogs with variable length. Each of them was recorded by a camera fixed on the back, and are divided in 10 different classes: car, drink, feed, look at right, look at left, pet, play with ball, shake, sniff and walk. Some of them have the same background, others were recorded in different places. The classes are strongly unbalanced, e.g. 25 videos for car and 10 videos for drink. To make them of equal length, a zero padding is inserted at the beginning of the shorter ones.
The UECPark instead is more regular, it is a unique video of a person doing exercises in a Park, such as running and climbing a rope. The video was divided into 766 fragment, 2 seconds long (120 frames), grouped into 29 classes. In this case too, the classes are unbalanced (one of them has just one label) and some of them are not so meaningful for the activity. In particular, the segments at the beginning and at the end of each activity.

5.3 Architecture

The general architecture is composed by a pretrained BNN feature extractor and an HD block for frames encoding. Without sufficient training data, such as in our datasets, the CNN is prone to overfitting, therefore it is essential to use the transfer learning, instead of training the CNN. The BNN is the same NiN-net described in the previous chapter, which was truncated at the second-last layer. Flattening the BNN output a 12288-bit HV is obtained for each input frame. The sequence of frame-HVs is encoded in n-grams, which are added all together at the end of the video. An HD n-gram can be generated in different ways: the most common one [5, 11] is $\rho(\rho C * B) * A = \rho\rho C * \rho B * A$ (fig. 5.1), where ρ indicates the permutation HD operator (sec. 2.1.3). The final video-HV is computed as:

$\sum_{i=0}^{\text{nframes}} x(i) * \rho x(i-1) * \rho\rho x(n-2)$. In this way, a set of frames is obtained for each video. During the inference, the similarity between the query and the video-HV would be proportional to the number of identical n-grams present in both the videos. To make an example the sequence ABCDE would be more similar to FGCDE than BADEC because it contains the ngram CDE. However if the dimension of the ngram had been 4 instead of 3, then the similarity would have been zero in both the cases, because neither ABCD nor BCDE were included in the set (FGCD, GCDE). So the length of the ngram should be carefully selected, usually the smaller the better (typically n=3).
Different kind of ngram were used. The ngrams can encode sequence of frames not strictly adjacent, for instance all the even frames can be skipped. This depends on the frames rate and on how much the video is frenetic. If the frame contest changes rapidly, then we need to encode all the frames, otherwise a subset of them should be selected. Indeed, if the contest changes too slowly we would encode a sequence of quite identical images.
Another kind of ngram with completely different properties is the following: $A + A * B + A * B * C$ where ABC is the sequence. This kind of encoding correlates the adjacent frames; A*B would generate an HV where all equal bits between A and B are set to 1, -1 otherwise, A*B*C instead would flip all the bits of C that was different in A and B. The first term instead, with a single letter, will generate the mean of all the frames when the final accumulation is performed. This kind of ngram will be the most suitable for our architecture because it computes the mean, which would identify all the videos with the same background, and other two terms which have a kind of feature extraction power.
Every video-HV belonging to the same class could be used to generate a class-HV in the AM. As an alternative, we can also use one prototype for each HV because of the number of samples is small and the AM would not be too large.

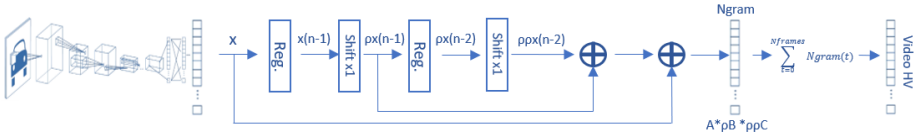

Figure 5.1. Firstly, each video frame is sent into the BNN which extracts a 12288-bit frame-HV. Secondly, the ngram is computed by means of two shift-register: $\rho\rho C * \rho B * A$, where the permutation ρ was implemented by shifting the HV of one position. Finally, the Video HV is obtained by accumulating all the frame HV belonging to the same video.

Integer ESN for HD

In [54] is suggested a method to directly binarize the ESN by using HD vectors, integer reservoir and clip activation function. The sequence of frames is encoded into the reservoir by using the shift (permutation) and bundling (sum) operations: $x(n) = f_k(Sh(x(n-1),1)+u^{HD}(n)+y^{HD}(n-1))$, where u^{HD} is the input projected in the HD space, Sh means shift opearation (which substitutes the expensive matrix multiplication with W^x) and f_k is the clipping activation function. Clipping threshold k limits the value range of activations (as well as the hyperbolic function in a typical ESN); all values over/under $\pm k$ are saturated to k. The readout matrix W_{out} is trained as a typical ESN with gradient descent, but we decide to use the HD perceptron instead.
In their case the sequence is composed by a sinusoidal signal, each $\sin(t)$ is quantized and associated to an HV in the item memory. So they are encoding a sequence of orthogonal HV kept from a limited alphabet. Our inputs are very different, coming from the BNN and are surely not orthogonal. However, it is worth to apply the architecture to our problem and see what is the result.

5.4 Analysis and results

Having a small dataset all the tests were repeated 10 times with different subdivisions of the samples between training set and test set. The results are shown in table 5.1 where each value is the mean of 10 tests. I found that a strong dropout

is necessary for this kind of dataset, because its restrict number of samples cause 100% overfitting of the training set after the first epochs. This would stop the HD perceptron training, remind that it takes some correction only in case of misprediction. The exact dropout percentage depends on the ngram encoding. It is suggested to increase this percentage until the training set requires an high enough number of epochs to reach the 100%. For our tests, we used values from 50% up to 99.9%.

ngram	Dog Acc.	Park Acc.
$A * \rho B * \rho\rho C$	40.1%	61.2
$A + A * \rho B * \rho\rho C$	54.4%	63.5
$A + A * B + A * B * C$	61.8%	56.5
$A + A * B$	63.3%	65.8%
intESN	45.7%	51.3%

Table 5.1. The simple A+A*B encoding results to be the best one in both the dataset.

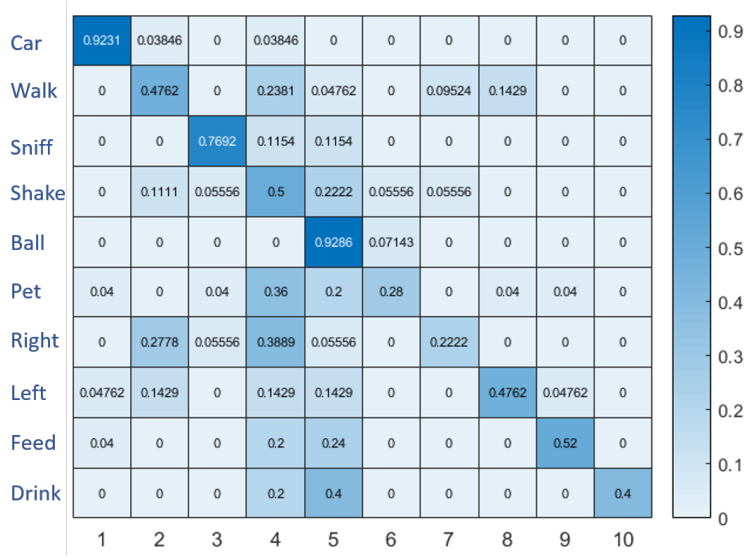

Figure 5.2. Confusion Matrix of the DogActivity classification.

Moreover, we found that if the classes stored in the Associative Memory are forced to be as far as possible, the result are slightly improved. This is obtained by performing this update at the end of each epoch:

$$AM[i] = AM[i] - AMavg * CosSim(AM[i], AMavg)$$

Therefore, each class is moved far away from the mean of all the classes, the closer the class, the farther it is sent. The effect is show in fig. 5.3. Of course, this update is done on the scalar AM, that one used during the HD perceptron which is then binarized before the inference. The best encoding is obtain with $x(n)+x(n)*x(n-$

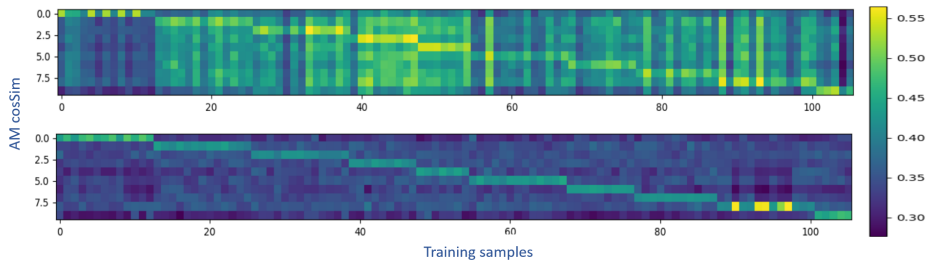

Figure 5.3. The scalar Associative Memory is updated by subtracting the mean of all the classes from each class, weighted by the distance of the class from the mean.

1) ngram type. The fig. 5.2 represents the confusion matrix of the best encoding. The classes which are better predicted are the ones with the same background and the results are comparable with [52]. The classes which particularly need the temporal information (turn right, turn left and shake) have a lower accuracy. The main contribution of the HD comes from the background recognition by performing a mean over all the frames. In fact, it classified correctly just the videos that are very similar, so with the same background. All the architectures that I tried have a too poor extraction power for timing information. The permutation (shift) used by HD for sequencing is typically applied on a fixed amount of items stored in an Item Memory. Indeed, HD sequencing works properly only if the frames which compose the sequence are present in different samples of the same class. In that case, only we can associate a frame with an item and get the expected results. Hence, what

we actually do in the front end is just encoding the incoming frames in a unique HV, sequencing them in Ngrams using shift-xor operator and accumulating them using the addition operator. Therefore, we can recall from the HV an already seen sequence, but nothing else, because this algorithm has not an intra-frames feature extraction power.
For instance, we can probably recognize the sequence ABCDE by training FGCDE because the same sequence CDE appears in both of them. However, if I have the same HV sequence CDE as output of the truncated BNN, this means that the input frames are very similar and so, in almost all the cases, they are recognized performing just the mean over all the frames of the same video. But for sure we cannot recognize two totally different sequence composed by different items. In the dataset of dog activity recognition I should recognize a dog that sniffs different places around the city. In that case the background is always different and the sequence changes totally for each sample. For this reason, all the methods used for encoding sequences mispredict this class. What we need is another feature extraction step which extracts information from the sequence of frames mainly related to the dog activity, e.g. in this paper [55] they used a GVD layer (which performs a mean over the feature sequence similarly to the HD) followed by a FCNN for the final classifciation. A direct HD encoding of the BNN output cannot work. We should get rid of the background and focus on the movements of the dog's head, probably the best way is a preprocessing on the input frames but it would be specific for this dataset, therefore not generalizable.
An alternative is to train a kind of Attention layer [56] able to recognize which features are really meaningful for classification and then using the HD for sequence encoding. This consideration comes from the fact that higher layers usually have categorical-level features, therefore probably it is possible to find that features which mainly refers to the dog position and discard all other features referring to the background. Without this approach the features related to the background would be predominant and the Hamming distance would be strongly affected, because in binary domain we cannot give more weights to some features with respect to others, therefore in those cases where the background is different we are practically giving a high weight to the noise instead of the useful information.

Chapter 6

Conclusion

The image recognition is a really challenging task which requires architecture with a powerful feature extractor. In the state of the art, the convolutional neural networks have almost no competitors. To apply sets of local filters generates a huge amount of low-level features; if they are properly selected and combined in the deeper layers, discriminant categorical-level features are produced. All this procedure require many fully precision operations which would consume too much power in an embedded device.

We explored the accuracy-complexity solution space with three architectures. The first one is based purely on HD computations in the binary space. It works for simple dataset such as the MNIST but it scales bad (35%) to the more challenging ones. More feature extractor power was necessary, therefore we decided to compute a Dictionary of filters learned by an unsupervised algorithm (kmeans) which projects the input patches into an high dimensional real space. We focused our efforts on the binarization of this space, looking for a distance preserving method; the most performant one is the thermometer code which maintains the same distortion generated by the quantization. Furthermore, a positional binding matrix encodes the position of each patch in the image. This architecture gives a 65% accuracy.

To further increase the accuracy, we should move towards some architectures more similar to the state of the art, as literature did in the last years. Different methods were proposed in order to binarize activations and weights of the CNNs, but they suffer from the binarization of the first and last layer. We propose a one-shot method, easy to use, applicable at any layer of a BNN. It can be directly applied to a pretrained BNN, it is able to binarize the last layer and it is useful for a fast analysis of the network during the design steps by highlighting which is the contribution of each layer to the classification. The maximum obtained accuracy is 83.7%. Removing all the layers but one, the accuracy drops to 58.6%, which is lower than the previous architecture where the Dictionary of patch filters works effectively as a convolutional layer, therefore the unsupervised training gives better result than the gradient descent. This means that the training of a BNN could be further improved.

The video activity recognition based on frame recognition and HD encoding did not give promising results because the HD is able to encode a sequence, but it cannot extract intra frames features related to the temporal evolution of the scene. The high accuracies are just given by an optimal recognition of the background which is different among the classes.

6.1 Future Works

In our first architecture, we used the HD for encoding the image directly into the HD space by using a clean-up item memory, where pixel position and colour shade codes are stored. This kind of setup is the usual one, all the HD applications are based on that because it exploits all the properties of the HD such as distributed information and robustness. In the other two architectures instead, the HD is applied just for classify binary HV generated by some feature extractor that does not maintain the same properties of a typical HD hypervector. There should be found a way to directly project the input pixel in the binary space, maintaining enough discriminant information, in order to extract them in the deeper layers. One possibility, which could fix the first BNN layer issue, is to feed the BNN directly with 10000 channels, which correspond to the HD encoding of each pixel.

The fully binary BNN convolutional layer would extract the correlation between the distributed HD binary features and a final fully connected layer would produce the final classification. This kind of architecture could really take advantage of the HD properties, avoiding the issues related to the binarization of scalar features and stepping closer to an efficient HW implementation for image recognition.

Bibliography

[1] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[2] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[3] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[4] O. Yilmaz, "Connectionist-symbolic machine intelligence using cellular automata based reservoir-hyperdimensional computing," *arXiv preprint arXiv:1503.00851*, 2015.

[5] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 64–69.

[6] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[7] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 2016, pp. 1–8.

[8] A. Rahimi, A. Tchouprina, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, pp. 1–12, 2017.

[9] M. A. Kelly, D. Blostein, and D. Mewhort, "Encoding structure in holographic reduced representations." *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, vol. 67, no. 2, p. 79, 2013.

[10] R. W. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059*, 2004.

[11] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.

[12] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 445–456.

[13] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[14] P. Datta and D. Kibler, "Symbolic nearest mean classifiers," in *AAAI/IAAI*, 1997, pp. 82–87.

[15] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[16] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[17] S. I. Gallant and P. Culliton, "Positional binding with distributed representations," in *Image, Vision and Computing (ICIVC), International Conference on*. IEEE, 2016, pp. 108–113.

[18] E. Weiss, B. Cheung, and B. Olshausen, "A neural architecture for representing and reasoning about spatial relationships," 2016.

[19] D. Wang and X. Tan, "Unsupervised feature learning with c-svddnet," *Pattern Recognition*, vol. 60, pp. 473–485, 2016.

[20] A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. J.

Wu, and A. Y. Ng, "Text detection and character recognition in scene images with unsupervised feature learning," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE, 2011, pp. 440–445.

[21] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[22] Scikit-learn. Kmeans. [Online]. Available: http://scikit-learn.org/stable/modules/clustering.html#k-means

[23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[24] A. Coates and A. Y. Ng, "Learning feature representations with k-means," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 561–580.

[25] The-Clever-Machine. The statistical whitening transform. [Online]. Available: https://theclevermachine.wordpress.com/tag/identity-covariance/

[26] ML-wiki. (2017) Bit sampling lsh. [Online]. Available: http://mlwiki.org/index.php?title=Bit_Sampling_LSH&oldid=760

[27] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[28] A. Blum, "Random projection, margins, kernels, and feature-selection," in *Subspace, Latent Structure and Feature Selection*. Springer, 2006, pp. 52–68.

[29] D. Needell, R. Saab, and T. Woolf, "Simple classification using binary data," *arXiv preprint arXiv:1707.01945*, 2017.

[30] Q. Gu, Z. Li, and J. Han, "Generalized fisher score for feature selection," *arXiv preprint arXiv:1202.3725*, 2012.

[31] Karpathy. Convolutional neural networks for visual recognition. [Online]. Available: http://cs231n.github.io/linear-classify/#softmax

[32] D. Cornelisse. An intuitive guide to convolutional neural networks. [Online]. Available: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[33] ujjwalkarn. An intuitive explanation of convolutional neural networks. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[34] M. Bernacki. Principles of training multi-layer neural network using backpropagation. [Online]. Available: http://home.agh.edu.pl/~vlsi/AI/

backp_t_en/backprop.html

[35] MathWorks. Introduction to deep learning: What are convolutional neural networks? [Online]. Available: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

[36] J. Yue-Hei Ng, F. Yang, and L. S. Davis, "Exploiting local features from deep networks for image retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 53–61.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[39] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[40] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[41] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[42] J. Yu, "Xnor-net-pytorch," https://github.com/jiecaoyu/XNOR-Net-PyTorch, 2017.

[43] A. G. Anderson and C. P. Berg, "The high-dimensional geometry of binary neural networks," *arXiv preprint arXiv:1705.07199*, 2017.

[44] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[45] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," in *Doklady Akademii Nauk*, vol. 114, no. 5. Russian Academy of Sciences, 1957, pp. 953–956.

[46] T. Xing, S. S. Sandha, B. Balaji, S. Chakraborty, and M. Srivastava, "Enabling edge devices that learn from each other: Cross modal training for activity recognition," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 2018, pp. 37–42.

[47] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[48] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *AAAI*, 2017, pp. 2625–2631.

[49] P. Jebashini, R. Uma, P. Dhavachelvan, and H. K. Wye, "A survey and comparative analysis of multiply-accumulate (mac) block for digital signal processing application on asic and fpga," *Journal of Applied Sciences*, vol. 15, no. 7, pp. 934–946, 2015.

[50] D. Graham, S. H. F. Langroudi, C. Kanan, and D. Kudithipudi, "Convolutional drift networks for video classification," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–8.

[51] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 659–686.

[52] Y. Iwashita, A. Takamine, R. Kurazume, and M. S. Ryoo, "First-person animal activity recognition from egocentric videos," in *International Conference on Pattern Recognition (ICPR)*, Stockholm, Sweden, August 2014.

[53] K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto, "Fast unsupervised ego-action learning for first-person sports videos," in *CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 2011, pp. 3241–3248. [Online]. Available: https://doi.org/10.1109/CVPR.2011.5995406

[54] D. Kleyko, E. P. Frady, and E. Osipov, "Integer echo state networks: Hyperdimensional reservoir computing," *arXiv preprint arXiv:1706.00280*, 2017.

[55] F. Scheidegger, L. Cavigelli, M. Schaffner, A. Malossi, C. Bekas, and L. Benini, "Impact of temporal subsampling on accuracy and performance in practical video classification," in *Signal Processing Conference (EUSIPCO), 2017 25th European*. IEEE, 2017, pp. 996–1000.

[56] E. Culurciello. The fall of rnn / lstm. [Online]. Available: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0