Development of a supervised learning algorithm through the use of Tensor Networks

Camilla Quaglia, Edoardo Antonaci, and Walter Zigliotto

ABSTRACT

The purpose of this work is to present a Matrix Product State (MPS) tensor network implementation for images recognition. In particular, the developed MPS algorithm was trained through the use of *gradient descent method* and was tested on three different datasets: the MNIST handwritten digits, the Fashion MNIST and a collection of pediatric chest X-ray images. The performance results are then reported through the graphic visualization of *accuracy* and *loss* behaviour during the training.
Moreover, the relationship between computation complexity scaling and tensor network's parameters variation is investigated. Finally, this study highlights how MPS is a promising tool to solve classical Machine Learning (ML) tasks, comparable to the state of the art ML algorithms, such as Neural Networks (NN).

INTRODUCTION

Tensor network methods have been applied in several field of physics and science, such as quantum information, quantum chemistry and condensate matter physics [1]. Moreover, tensor networks are also increasingly finding applications in machine learning [2] [3]. For instance, Stoudenmire and Schwab [3] implemented a Matrix Product State (MPS) algorithm to classify the MNIST dataset, using the Density Matrix Renormalization Group (DMRG) technique to train the model. Further, Cavinato et al. [4] mapped the radiotherapy dose optimization problem into an Ising Model ground state search, solved through a Tree Tensor network algorithm.

In parallel with the growth of tensor network application, some APIs are becoming useful to accelerate research in that field. Among the others, *TorchMPS* [5], a framework to work with MPS models with Pytorch, and *TensorNetwork* [6], a library that acts as a tensor network wrapper for *TensorFlow*, *JAX*, *PyTorch*, and *Numpy* to implement such algorithms, are the most employed toolbox for the Python programming language.

In this work, the kind of *ansatz* applied for tensor networks is the MPS, a popular technique in quantum physics that we used to develop a supervised learning algorithm, aimed at images classification.
In particular, three kind of dataset were taken into account: MNIST, Fashion MNIST and Chest X-ray images from pediatric patients. Precisely, the first two datasets were chosen to have a consolidated and well-known benchmark to evaluate the algorithm performance, while the third file was selected to try a possible healthcare MPS algorithm application in order to obtain a rapid pneumonia diagnosis, which is fundamental nowadays to reduce children mortality. In fact, according to the World Health Organization (WHO), for children under 5 years old pneumonia provoked the 15% of all deaths in 2017 [7].

From a practical point of view, the *Tensorflow* library, which is familiar in the ML framework, turned out to be handy to solve the task. In particular, the *Tensorflow* version *2.4.1* on *Python3* is adopted through Google Colab, since it allows us free GPU access.
By means of the aforementioned tools, this work presents the potentiality of tensor networks applied to a supervised learning problem.
Firstly, in the Theory section, the general MPS theoretical aspects are presented. Moreover, a detailed explanation of the datasets and the preprocessing chain is reported in the Method section, together with the MPS algorithm description. Finally, the results are presented along with a brief discussion on the possible future steps of this algorithm.

To summarize, the most promising MPS Tensor Network results achieved were:

- 97% test accuracy classification for the MNIST dataset.
- 87% test accuracy classification for the Fashion-MNIST dataset.
- 95% test accuracy for the normal and pneumonia identification through the pediatric X-rays images.

THEORY

In this section, the main theoretical concepts behind this work are summarized [1]. The framework in the description is the quantum mechanics, even if the tools are applied in our ML task.

As it is known, a N-rank tensor is an object with N indexes of the form

$$T_{\alpha_1,\alpha_2,.....,\alpha_N} \tag{1}$$

that can be graphically represented as a body with N legs, one for each index, like the figure 1 shows. Low rank tensors are well known, as scalars (tensors of rank 0), vectors (tensors of rank 1) and matrices (tensors of rank 2).

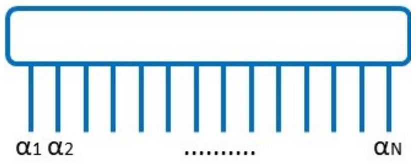

FIG. 1: A tensor of rank N

A tensor network is a network composed by the aforementioned objects, connected through mathematical operations, such as the contraction of different indexes. The latter is the generalization of the matrix-vector multiplication rule, where we can apply the Einstein's notation for a more compact form.

$$\hat{A}|\psi\rangle = \sum_{\beta} A_{\alpha,\beta}\psi_\beta \underset{a}{=} A_{\alpha,\beta}\psi_\beta \underset{b}{=} C_\alpha \tag{2}$$

where in a the Einstein's notation is adopted and in b the contraction is computed.

Tensors are nothing else than a way to organize information. An important point to highlight is that one could reshape, for example, a four dimensional vector in a 2×2 matrix and vice versa, without losing information. Generally, a rank-n tensor can be rearranged in a tensor of a different rank, if the operations acting on the objects change accordingly. In the case of matrix reshaping into a vector, two indexes are fused into one and this *index fusion* operation can be generalized as follow

$$\alpha_1......\alpha_N = d - coding(\mathbf{i}) \tag{3}$$

where we have N indexes α_i, each one can assume d values, and the operation is given by the d-nary coding of the global index $\mathbf{i}$.
The *index fusion* operation allows the **compression** of information. In particular any tensor of rank N can be transformed in a matrix: we have to choose a bipartition of the N indexes and *fuse* together the indexes in each of the two groups. The result is the following

$$T_{\alpha_1,\alpha_2,.....,\alpha_N} = T_{\mathbf{i},\mathbf{j}} \tag{4}$$

At this point one could think of diagonalize that matrix, or generally perform the *Singular Value Decomposition (SVD)*. This is a way to compress our e.g. many body wave function, to find its ground state.
Another way to do that is through the *Matrix Product State (MPS)*, a kind of tensor network that include information of quantum correlations/entanglement, on the contrary of another tensor "network" that is the *Mean Field (MF)*, of which the MPS is a generalization.

The MPS is defined as

$$|\psi_{MPS}\rangle = \sum_{\vec{\alpha},\vec{\beta}} A^{\beta_1}_{\alpha_1} A^{\beta_1\beta_2}_{\alpha_2} ....A^{\beta_{N-1}}_{\alpha_N} |\alpha_1, \alpha_2....\alpha_N\rangle \tag{5}$$

Therefore, we have N sites α_i, with $\alpha_i = 1, ..., d$, each one corresponds to one tensor A_{α_i}, and $\beta_j = 1, ..., m$, where m is called the *bond dimension*. The latter indexes are *a plus* with respect to the *Mean Field (MF)*, to add correlation. Now, if $m = d^N$ we would have an exact many body state and if $m = 1$ we have the *MF* approximation. In fact, the *MPS* could be considered as a trade off between the two extremes.
Graphically the *MPS ansatz* is reported in figure 2, for N=5.

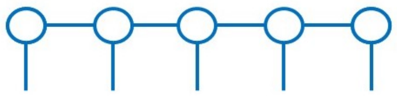

FIG. 2: *MPS ansatz* for N=5

METHODS

In the theory section is depicted that the tensor networks are a powerful tool to act on a high dimensional space. In our task this space consists of N greyscale pixels of an image. In the case of MNIST and Fashion-MNIST datasets $N = 28^2 = 784$, while for the pediatric pneumonia file N was fixed at $128^2 = 16384$. In the following part, the operative procedure to manage these vectors is described along with the algortihm description.

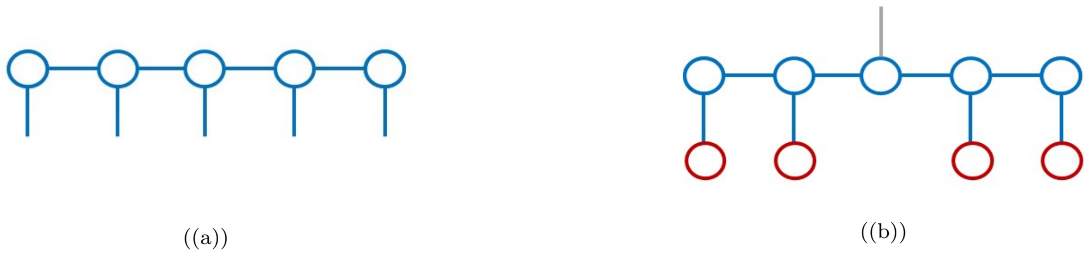

FIG. 3:
(a) MPS tensor network. The free indexes represent the pixels in the image
(b) The inner product between the *variational* MPS (in blue) and the 2-dimensional vectors images (in red)
. The operation is analytically expressed by the equation (12). All the indices in the MPS, that were free in figure 3(a), are contracted with the images vectors, except the index l, in lighter gray, that represents the label.

Preprocessed data for Tensor Networks

Each image we want to classify is represented by a 28×28 or a 128×128 matrix, as previously stated. In particular, for the X-rays dataset [8], the size dimension of the original image has to be reduced in order to optimize the computation efficiency.

To manage these matrices in order to pass them through the developed one-dimensional tensor network, each image is:

- Flattened, which means that a dimension is thrown away (e.g. reshaping the 28×28 matrix on a vector of 784 elements).
- Normalized, to optimize the efficiency of function in a restricted range.

We point out that in this way the two-dimensional nature of the image is lost.
However, our purpose is just to explore the application of the tensor network technique in the machine learning world, and even our implementation revealed good. A more precise approach in image classification problems can be a two-dimensional tensor network [9].
In second place, each pixel value of the so transformed image is encoded into a 2-dimensional vector space through the use of a local feature map. Therefore, each flattened image composed by N pixels is mapped in a 2^N-dimensional vector space.

In particular an example of two commonly used local feature maps is reported here: the first one is

$$\Phi(x) = \begin{pmatrix} 1-x \\ x \end{pmatrix} \tag{6}$$

called *linear feature map* while the second one is

$$\Phi(x) = \begin{pmatrix} \cos\left(\frac{\pi x}{2}\right) \\ \sin\left(\frac{\pi x}{2}\right) \end{pmatrix} \tag{7}$$

where x is the normalized pixel value ($x \in [0, 1]$). During this study, only the first one was adopted.
The tensor product of the Φ defined above represents the total image space, which has dimension 2^N:

$$\Phi^{s_1 s_2 \dots s_N}(\vec{x}) = \Phi^{s_1}(x_1) \otimes \Phi^{s_2}(x_2) \otimes \dots \otimes \Phi^{s_N}(x_N) \tag{8}$$

In our case, the general s_j, with $j = 1, ...N$, can assume only two possible values, 0 or 1. The components of the "images tensor" defined in (8) as $\Phi^{s_1 s_2 \dots s_N}(\vec{x})$ are

$$\Phi^{s_1}(x_1)\Phi^{s_2}(x_2)...\Phi^{s_N}(x_N) \tag{9}$$

The number of components is therefore 2^N.
The kind of "image tensor" in the image space we consider is the MPS tensor network. It has as before 2^N components and in our task can be written in the following way, with evident analogy with the general expression in equation (5)

$$T_{s_1,s_2,\dots,s_N} = \sum_{\vec{\alpha}} A^{(1)}_{s_1\alpha_1} A^{(2)}_{s_2\alpha_1\alpha_2} A^{(3)}_{s_3\alpha_2\alpha_3} \dots A^{(N)}_{s_N\alpha_N} \tag{10}$$

where the α_k indexes, with $k = 1, ..., N$, can vary in a range, called the *bond dimension* χ, which is an hyperparameter of our model. The *bond dimension* determine the size of the A tensors. Moreover, the A tensors coefficients are the *weights* or *variational parameters*, that are fixed via training procedure.

Dataset description

In this section the datasets on which our code is tested are described.

MNIST dataset. This dataset contains 28×28 images of 10 handwritten digits in grayscale. The aforementioned 10 digits are the numbers from 0 to 9. The images were loaded from the `Keras` page `https://keras.io/api/datasets/mnist/`. In figure 4

Label	*Description*
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

TABLE I: Fashion-MNIST images and corresponding labels

an example of an image in this dataset. In particular the units are partitioned in 60000 samples as training set and 10000 as test set.

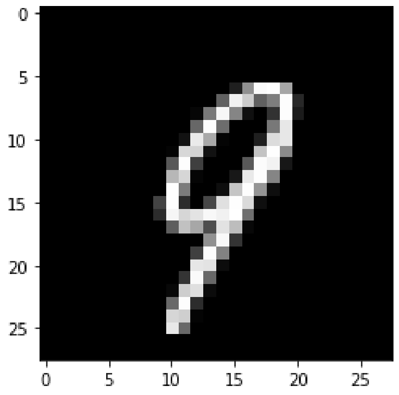

FIG. 4: An example of figure in the MNIST dataset.

Fashion MNIST dataset. This dataset contains 28×28 images of 10 different kind of clothes, again in grayscale. As in the MNIST dataset, the images can have 10 labels, listed in table I. The images were loaded from the `Keras` page `https://keras.io/api/datasets/fashion_mnist/`. In figure 5 an example of an image in this dataset. Also here the units are partitioned in 60000 samples as training set and 10000 as test set.

Chest X-Ray Images. The dataset consists of 5863 chest X-ray images (JPEG format) of pediatric subjects of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. It is labelled into 3 possible classes:

- Normal.
- Bacterial pneumonia.
- Viral pneumonia.

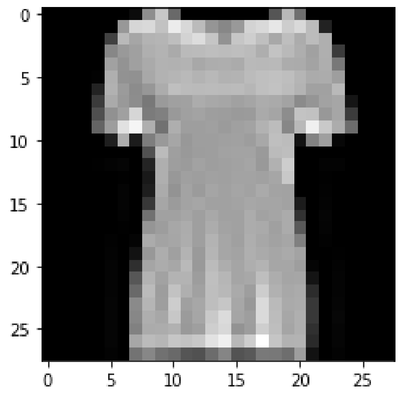

FIG. 5: An example of image in the Fahion-MNIST dataset.

The images were loaded from `https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia`. Figure 6 is an example of pneumonia viral lung, where at the left side there is the original image (that in general can have variable shape) and at the right side the resized 128×128 image.

Objective function

In general in a classification problem the target is a map f from the space of *objects* to the space of *labels*. For example, in the MNIST and Fashion-MNIST datasets the *objects* are respectively handwritten digits and clothes images and the *labels* are the digits from 0 to 9. To *update* the map f, in the machine learning framework, pairs of labeled examples are exploited. In our task the aforementioned pairs are of the type ($\mathbf{x}$,y), where $\mathbf{x}$ are the flattened images, and y the corresponding labels.

$$\mathbf{x} = (x_1, x_2, ..., x_N) \qquad y \in \{0, 1, 2, ..., L-1\} \tag{11}$$

where x_j is the normalized pixel. In the case of the MNIST and Fashion-MNIST datasets $N = 28^2$ and $L = 10$.
The classification map f, exploiting the *MPS* framework, is defined as the inner product between the 2-dimensional images vectors, defined above, and the *variational* MPS as follow

$$f^{(l)}(\mathbf{x}) = \sum_{s_1,..,s_N=0}^{1} T^l_{s_1,..,s_N} \Phi(x_1)_{s_1} ... \Phi(x_N)_{s_N} \tag{12}$$

The inner product in equation (12) can be represented graphically as in figure 3(b). The label l, that can assume values from 0 to 9 (e.g., MNIST dataset), has an arbitrary initial position in the chain. Operatively, we assign at the label l the middle of the chain, so $N/2$,

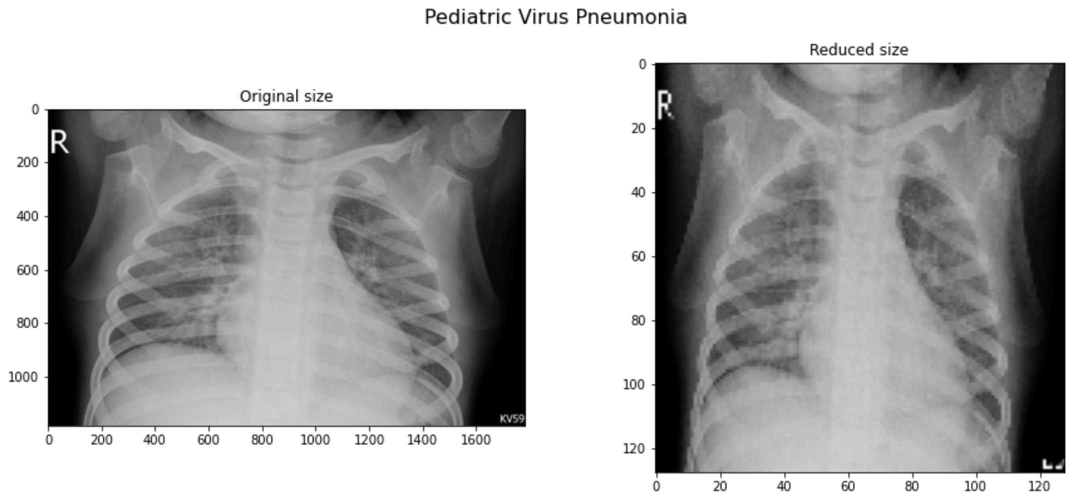

FIG. 6: An example of pediatric viral pneumonia through lung X-rays. On the left side the original image, on the other hand the 128×128 reshape image.

as initial position. The purpose of this choice will be clarified in the next section.

The classification map f is than selected such that for each image the corresponding label gives the maximum *superimposition* between its MPS and the corresponding image vector. Matematically,

$$f(\mathbf{x}) = \underset{l}{argmax}\, f^l(\mathbf{x}) \tag{13}$$

According to the typical machine learning procedure, the components of the A tensors (equation (10)), namely the *variational parameters* or *weights* should be tuned in order to minimize an objective function in the training set.
As objective function, the multi-class cross-entropy was chosen. The latter is a commonly used loss function for classification tasks.
It is defined as follows, where $\mathcal{D}$ is our training set.

$$CE = -\sum_{(\mathbf{x}_i, y_i)\in\mathcal{D}} log\ softmax f^{(y_i)}(\mathbf{x}_i) \tag{14}$$

Where $softmax$ is an activation function. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. Its definition is

$$softmax f^{(y_i)}(\mathbf{x}_i) = \frac{e^{f^{(y_i)}(\mathbf{x}_i)}}{\sum_{l=0}^{L-1} e^{f^{(l)}(\mathbf{x}_i)}} \tag{15}$$

Looking at the formula above becomes clear to interpret the output of this function as the *predicted probabilities* for each label assigned to an object. In simple words, since a decreasing behaviour of the *Cross Entropy* in equation (14) is expected (ideally *Cross Entropy* $\to 0$), the final prediction corresponds to the label with maximum probability.

Code development and Optimization

Firstly, the listing 1 shows the practical procedure used to encode data in the tensor network, as described in the above *Method* section. The aforementioned piece of code belongs to the *Python* script `MPS_utils.py`, where the algorithm for this work is contained.

Listing 1: Data Encoding for MNIST and Fashion MNIST

```
# DATASETS PREPARATION for MINST and FashionMNIST
#####################################################
def dataset_preparation(train_set, train_lab,
                        test_set, test_lab):
  # Function to prepare the images to be given
  # as input to our MPS algorithm
  #
  # Args
  # train_set: raw training set images
  # train_lab: labels of the training set
  # test_set: raw test set images
  # test_lab: labels of the test set
  #
  #
  # Return
  # fin_train_set: prepared training images
  #                (2-dim vectors)
  # one_hot_train: training labels in one hot encoding
  # fin_test_set: prepared test images
  #                (2-dim vectors)
  # one_hot_test: test labels in one hot encoding

    max_val = np.max(train_set)  # normalize pixels
    min_val = np.min(train_set)
    delt_val = max_val - min_val

    norm_train_set = (train_set-min_val)/delt_val
    norm_test_set = (test_set-min_val)/delt_val
    ##########################################

    # flattening images
    reshape_train = list(norm_train_set.reshape(
    len(norm_train_set), -1))
    reshape_test = list(norm_test_set.reshape(
    len(norm_test_set), -1))

    def psi(x):
        x = np.array((1 - x, x))     # linear map
        return x

    def mapping_dataset(dataset):
        mapped_dataset = []

        for jj in range(len(dataset)):
            psi_map = list(map(psi, dataset[jj]))
             # apply map to each pixel
            mapped_dataset.append(psi_map)

        return np.array(mapped_dataset)

    fin_train_set = mapping_dataset(reshape_train)
    fin_test_set = mapping_dataset(reshape_test)

    # commute labels in one hot encoding format
    ##########################################
    def one_hot_label(labeled_data):
        n = max(labeled_data) + 1
        one_hot = np.zeros((len(labeled_data),n))

        for ii in range(len(labeled_data)):
            one_hot[ii][labeled_data[ii]] = 1

        return one_hot

    one_hot_train = one_hot_label(train_lab)
    one_hot_test = one_hot_label(test_lab)
    ##########################################

    return fin_train_set, one_hot_train,
                fin_test_set, one_hot_test
```

The development of the MPS algorithm can be divided in two parts:

- **Forward pass**, that correspond to calculate the inner product defined in equation (12), i.e. contract the tensor network. In other words, it is the calculation of the $f^{(l)}(\mathbf{x})$ for a given input $\mathbf{x}$.
- **Backward pass**, for which the *Tensorflow* library turned out to be handy, as specified in the following section, allowing the automatic computation of gradient descent along with the training of the model.

A general procedure to perform the forward pass described above is the "sequential contraction" of the MPS, as described in figure 7. The contraction is performed "from left to right" as the figure clearly shows. We call d the size of the horizontal legs in the chain and χ the *bond dimension*, so the size of the vertical legs. In our problem d denotes the dimension of the pixel space, so $d = 2$, and χ is kept constant. With this notation, the cost of contracting an horizontal leg is $\mathcal{O}(d\chi)$ and the cost of contracting a vertical leg is $\mathcal{O}(d\chi^2)$. Going ahead with the contraction at a certain point we will encounter the label l. We have to take into account this factor in the cost. If we have L different labels ($L = 10$ in the MNIST and Fashion-MNIST dataset) the total cost is of order $\mathcal{O}(NLd\chi^2)$. A way to avoid the extra factor L is to start the contraction from the extremes of the chain and contract with the tensor that carries the label index as final step. Recall that the position of the latter is chosen at $N/2$. This procedure allow to obtain a total cost of order $\mathcal{O}(Nd\chi^2)$.
The contraction we follow for our implementation is inspired by the paper [2] and can be defined as a parallelized MPS contraction with respect to the sequential one, in figure 7. The procedure for this parallel contraction is described in figure 8. One can see that this method requires matrix-matrix multiplications, while the above contraction requires only matrix-vector contractions. Therefore, the cost scales as χ^3, being higher for this second method. However, the advantage is that each step is easy to parallelize as the matrix-matrix multiplications are independent from the results of the previous (neighbouring) steps. Effectively, this method were preferred since the batching operations supported by `TensorFlow` allows a straightforward parallel implementation.
In the listing 2 the contraction implementation ispired by the figure 8 is shown.

Listing 2: The contraction implementation

```
def contraction(lor_side):
     # MPS contraction
    dim = int(lor_side.shape[0])
    while dim > 1:
        half_dim_it = dim // 2
        old_dim = 2 * half_dim_it
        remaining_side = lor_side[old_dim:]
        lor_side=tf.matmul(lor_side[0:old_dim:2],
                       lor_side[1:old_dim:2])
        lor_side = tf.concat([lor_side,
                       remaining_side], axis=0)
        dim = half_dim_it + int(dim % 2 == 1)
    return lor_side[0]
```

In our work the optimization of the tensor net-

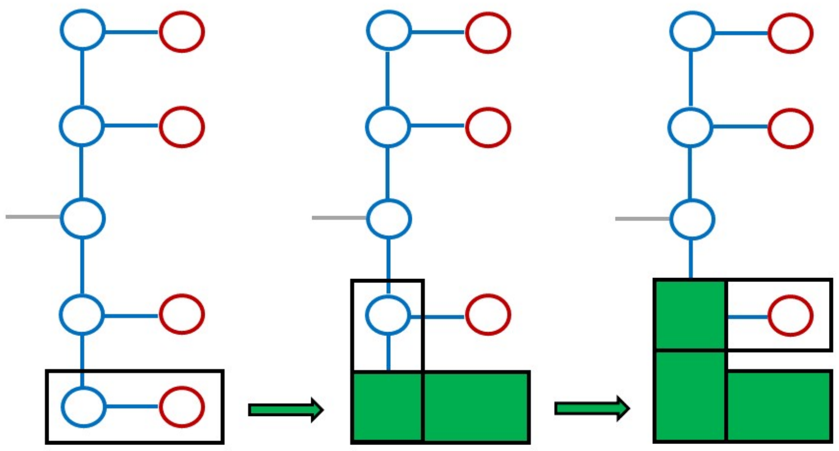

FIG. 7: The sequential contraction of a MPS

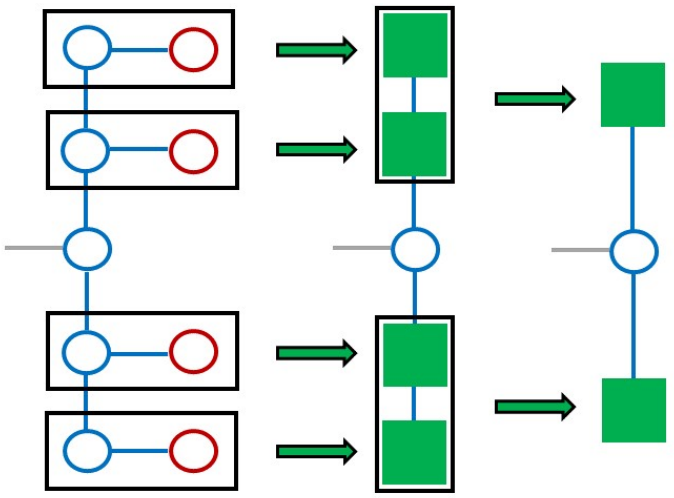

FIG. 8: From left to right: Parallel contraction of all the pixel vectors (red) with the variational MPS. The total cost of this operation goes as $\mathcal{O}(Nd\chi^2)$
Contraction in pairs, parallel once again, of the tensors created at the step before.
Repeated contraction in pairs and parallel until the chain is fully contracted.

work has been done through the gradient of the objective function, as [2]. Indeed `TensorFlow` calculates the gradient of the loss function in an automatic way. The objective function we have chosen is the multi-class cross entropy, written in equation (14). In the code, the function is expressed by the command `tf.keras.losses.categorical_crossentropy`, available in `TensorFlow 2.x`.
The calculated gradients are then used to minimize the loss via a *stochastic gradient descent* procedure. For this purpose, the `tf.keras.optimizers.Adam` was adopted. This kind of optimization method was preferred with respect to sweeping algorithms, such as DMRG, considering the more straightforward implementation in a machine learning library as `TensorFlow`.

Another feature of the implemented method is that the *bond dimension* χ, the only MPS hyperparameter we have, is set a priori and is kept constant during training. On the contrary in the sweeping method, a singular value decomposition (SVD) step allows to change the parameter during training. However we found that the results do not depend on the value set, for $\chi > 10$, as shown in the result section.

RESULTS

The results are obtained running the notebooks `Test_MNIST_bond.ipynb`, `Test_MNIST.ipynb`, `Test_Fashion.ipynb`, and `test_X-rays_v1_3.ipynb` on

Google Colab, a cloud platform provided by Google. The choice of proceeding in this way was dictated by the fact that *Colab* allows to have free access to a *GPU* hardware. The latter speeds up our training procedure in a considerable way with respect to a local running on our laptops. In this section the training histories of the three different datasets described above are shown.

MNIST dataset. In figure 9 the results for accuracy and loss reached by our algorithm on this dataset are presented. As optimizer, *Adam* is found to be an efficient option. During the training, the batch size is fixed to 10, while the number of epochs to 100. We point out that the results do not depend on the particular bond dimension χ, as the figure 10 states. The latter is obtained with reduced training and test sets, respectively 6000 and 1000 images, with respect to the total MNIST dataset, used to produce figure 9. The choice was dictated by two main reasons:

- The computational time, too huge for the *Google Colab* runtime using the whole dataset.
- The capability to show qualitatively that the bond dimension set does not affect the results, even with a smaller dataset.

Instead the learning rate chosen plays a role in the final results, as a machine learning practitioner could guess. After empirical trials to find the "best" learning rate, in this case 10^{-4} turned out to be a good option and so this value is fixed during training. To conclude, the performance of the algorithm was excellent, in fact the algorithm reached a 97% of accuracy in the test set.

Fashion MNIST dataset. In figure 11 the results for accuracy and loss achieved by the MPS algorithm on this dataset are reported. As optimizer, *Adam* is again exploited and the batch size is fixed to 10 while the number of epochs to 100, as previously done. Given that the results do not depend on the particular bond dimension χ, in this instance it is kept fixed to 10. The learning rate is fixed to 10^{-4}, found to be an optimal values. Finally, the performance of the algorithm was acceptable, with a 87% of accuracy in the test set. Indeed this dataset was more complex than the previous one, probably due to the similar shape of different clothes among each other.

Chest X-Ray Images. The main purpose of this part was to distinguish between two set of classes. In order:

- Normal vs Pneumonia lungs.
- Viral Pneumonia vs Bacterial Pneumonia lungs.

For both the classification tasks, the bond dimension was fixed at 20, the batch size at 10, the number of epochs at 100 and *Adam* was chosen as optimizer for the gradient descent. In particular, for the latter, the learning rate was set at 0.00001 and $\hat{\epsilon}$, a numerical stability constant [10], at 0.008.
On one hand, according to figure 12 accuracy and loss values for X-ray pediatric pneumonia and normal lung identification are reported. The maximum accuracy achieved on the test set is 95%. This is comparable with the state of the art ML algorithm.
On the other hand, figure 13 presents worse performance. This is probably due to the complexity of the task and to the lower dimension of the dataset. Furthermore, a tuning of the hyperparameters could be useful, as we adopt the same of the previous task. Nevertheless, the MPS algortihm was able to classify at a maximum of 75%.

As we can see from figure 11 and figure 10 (in particular for $\chi = 10$) the test loss behaviour is unstable and tends to increase even if the test accuracy is converging at a fixed value with a stable behaviour. This trend is probably motivated by the presence of few outliers in the test set, which are misclassified and make the loss becoming unstable. Despite this behaviour, our model is still learning on the rest of the dataset, as the accuracy states. The problem is linked to the use of the *cross entropy* as loss, which is not a *bounded* loss.

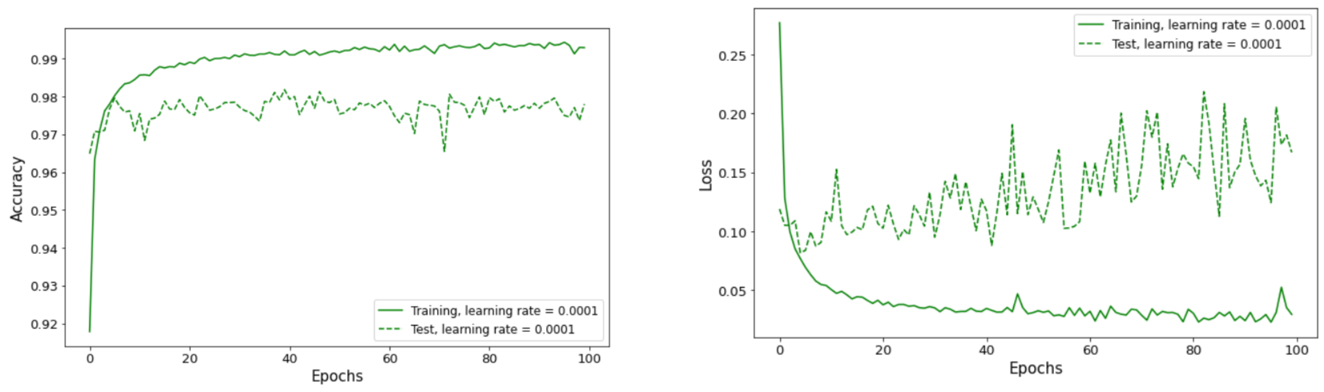

FIG. 9: Accuracy (left) and loss (right) reached on MNIST dataset.
The training and the test set are composed respectively of 60000 and 10000 images. For this plot the bond dimension is fixed to 10 and the learning rate to 10^{-4}.

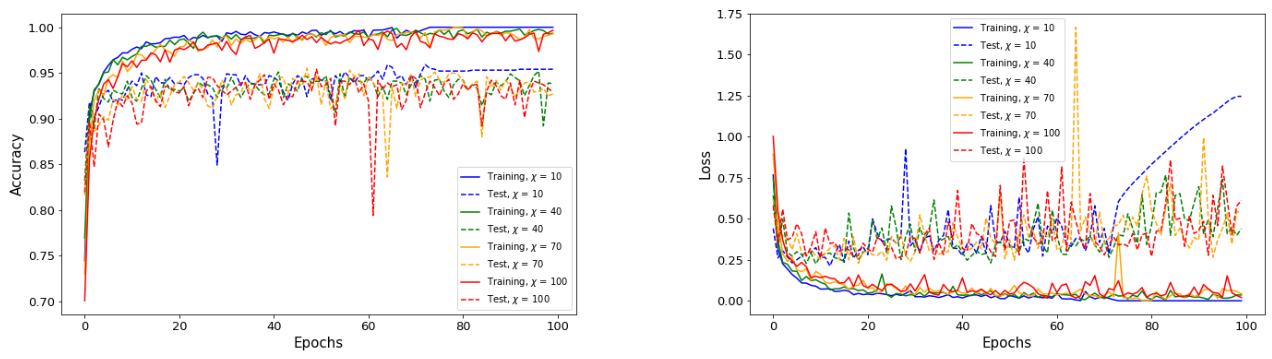

FIG. 10: Accuracy (left) and loss (right) reached on MNIST dataset, for different bond dimensions χ. In this case the training and the test set are composed respectively of 6000 and 1000 images.
The plots show that the results are independent from the χ values.

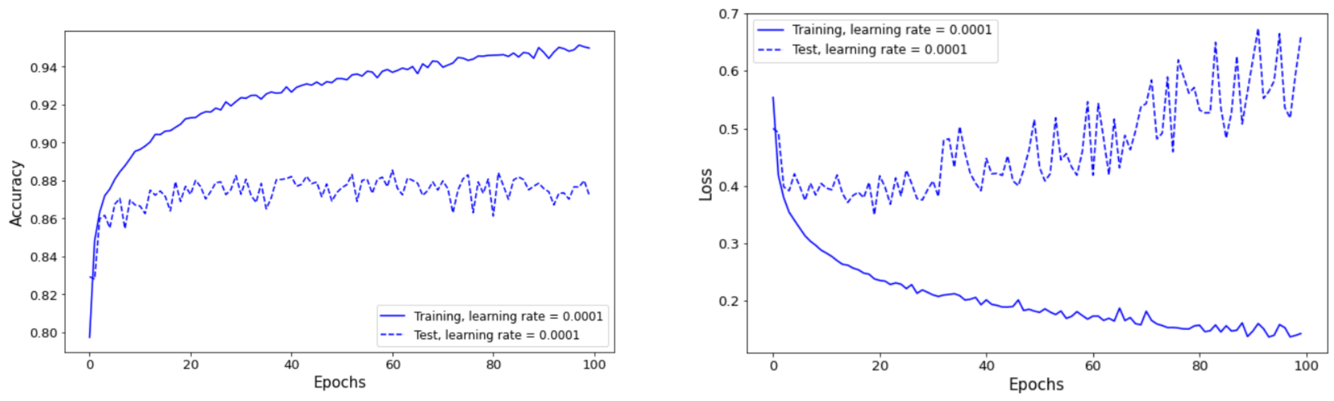

FIG. 11: Accuracy (left) and loss (right) reached on Fashion-MNIST dataset.
The training and the test set are composed respectively of 60000 and 10000 images. For this plot the bond dimension is fixed to 10 and the learning rate to 10^{-4}.

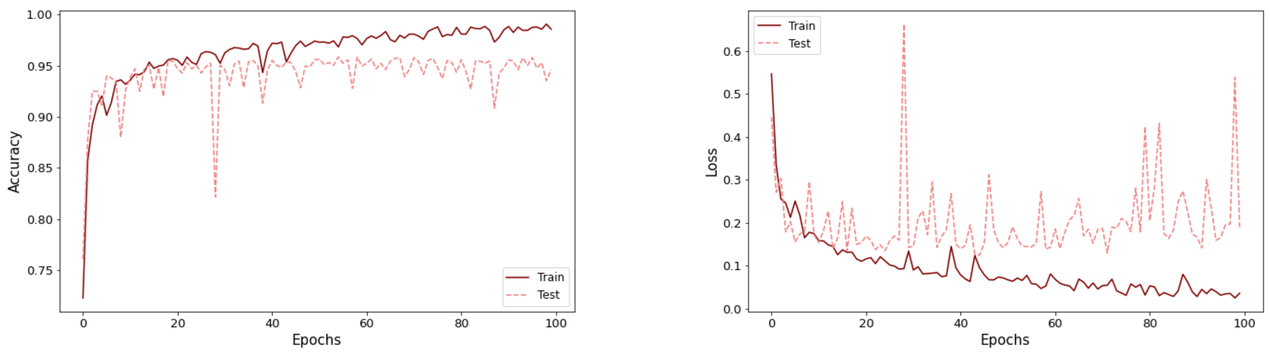

FIG. 12: Accuracy (left) and loss (right) reached on Chest X-ray dataset, bond dimensions $\chi = 20$.

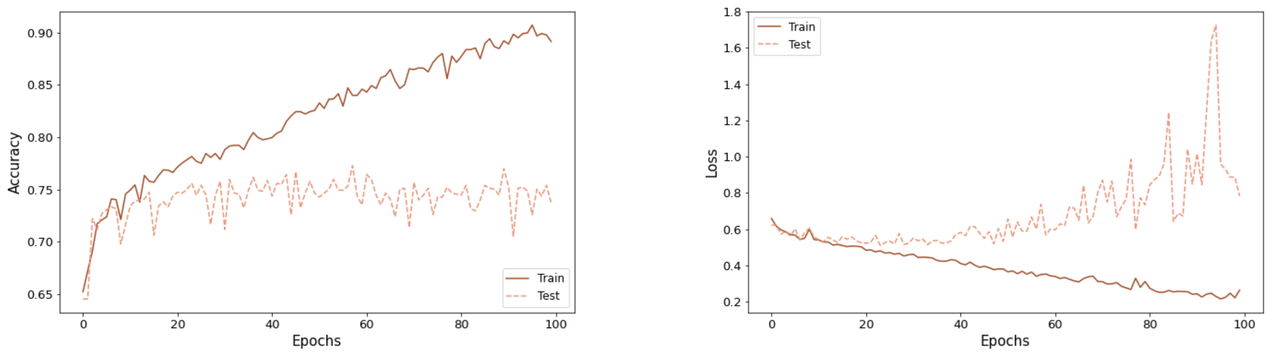

FIG. 13: Accuracy (left) and loss (right) reached on Chest X-ray dataset, bond dimensions $\chi = 20$.

CONCLUSIONS

The main focus of this work was to implement a surpervised learning algorithm through the use of MPS tensor network. Practically, the contraction method described in figure 8 was exploited and to train the model the use of the `Python` library `TensorFlow` turned out handy. We optimize the training using the built-in automatic gradient procedure, selecting *Adam* as optimizer with a low learning rate. A limitation encountered during this work regards the hardware acceleration. To face the problem graphics processing units (GPU) are exploited. However, as a future step, others solutions, as clusters of GPUs (i.e., general purpose processors (GPGPU)) could be perceived. As example, we can cite the *NVIDIA CUDA* platform for GPGPU programming, that allows to perform parallel computations, or even the open parallel programming standard *OpenCL*.

The results of the classification on MNIST and Fashion MNIST datasets were satisfying, reaching respecrively a 97% and 87% test accuracy. As far as concerns the X-ray dataset analysis, a test accuracy of 95% was achieved distinguishing between pneumonia and normal lungs. Nevertheless, more studies have to be conducted to increase the performance of the algorithm, in particular for the last classification task. By the way, Kermany at al. [8] achieved an accuracy of $\sim$ 93%, for the first classification topic and the accuracy for the Viral pneumonia vs Bacterial pneumonia was slightly under 91% . Moreover, it is important to highlight that the MPS algorithm runs over reduce size images of just 128 $\times$ 128 due to memory allocation problem. Therefore, some improvements can be reached hosting the same algorithm on a more powerful machine. Probably, in this last scenario, TPU (Tensor Processing Unit) could become a natural tool to achieve better performances, accelerating all the operations that regards tensor network framework [11]. Furthermore, the development of an automatic grid search for the best hyperparamters identification could improve the accuracy.

Finally, another future attempt to investigate is the development of higher dimensional models. As an example, Tree Tensor Network (TNN) is thought to suit more the two dimensional nature of images. Such improvements could better fit the datasets analysized and even more complex ones (e.g., the Stabilo Digitpen file [12], that has high dimensional input data).

To conclude, this project is an example of the power of the tensor network method. In particular, the fusion of this tool with a machine learning framework, effectively in the `TensorFlow` library, allows to solve classification problems, along with performance already comparable to more traditional machine learning methods, such as neural networks.
All of the code used to produce the results in this work is available at the following GitHub repository [13].

REFERENCES

[1] Simone Montangero. *Introduction to Tensor Network Methods Numerical simulations of low-dimensional many-body quantum systems.* Springer International Publishing, 2018. URL: https://doi.org/10.1038/s41598-020-78652-0.

[2] Stavros Efthymiou, Jack Hidary, and Stefan Leichenauer. "Tensor Network for Machine Learning". In: *CoRR* abs/1906.06329 (2019). arXiv: 1906.06329. URL: http://arxiv.org/abs/1906.06329.

[3] E. Miles Stoudenmire and David J. Schwab. *Supervised Learning with Quantum-Inspired Tensor Networks.* 2017. arXiv: 1605.05775 [stat.ML].

[4] Samuele Cavinato et al. *Optimizing Radiotherapy Plans for Cancer Treatment with Tensor Networks.* 2020. arXiv: 2010.09552 [physics.med-ph].

[5] Jacob Miller. *TorchMPS.* https://github.com/jemisjoky/torchmps. 2019.

[6] Chase Roberts et al. "TensorNetwork: A Library for Physics and Machine Learning". In: (May 2019). arXiv: 1905.01330 [physics.comp-ph].

[7] *Pneumonia Available at https://www.who.int/news-room/fact-sheets/detail/pneumonia.* URL: https://www.who.int/news-room/fact-sheets/detail/pneumonia#:~:text=Pneumonia%20accounts%20for%2015%25%20of,and%20by%20addressing%20environmental%20factors.. (accessed: 02.08.2019).

[8] Daniel S. Kermany et al. "Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning". In: *Cell* 172.5 (2018), 1122–1131.e9. ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2018.02.010. URL: https://www.sciencedirect.com/science/article/pii/S0092867418301545.

[9] Ding Liu et al. *Machine Learning by Two-Dimensional Hierarchical Tensor Networks: A Quantum Information Theoretic Perspective on Deep Architectures.* 2018. URL: https://openreview.net/forum?id=ryF-cQ6T-.

[10] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[11] *Tensor Processing Unit.* URL: https://cloud.google.com/tpu/docs/system-architecture#hardware_architecture.

[12] *Stabilo Digipen.* URL: https://stabilodigital.com/onhw-dataset/.

[13] *MPSgithub.* URL: https://github.com/walterzigliotto/MPS_images_classification.