algorithms

Article

LSTM Accelerator for Convolutional Object Identification

Alkiviadis Savvopoulos [1], Andreas Kanavos [1,*], Phivos Mylonas [2] and Spyros Sioutas [1]

[1] Computer Engineering and Informatics Department, University of Patras, Patras 26504, Greece; asavv@ceid.upatras.gr (A.S.); sioutas@ceid.upatras.gr (S.S.)
[2] Department of Informatics, Ionian University, Corfu 49100, Greece; fmylonas@ionio.gr
* Correspondence: kanavos@ceid.upatras.gr

Received: 4 August 2018; Accepted: 12 October 2018; Published: 17 October 2018

Abstract: Deep Learning has dramatically advanced the state of the art in vision, speech and many other areas. Recently, numerous deep learning algorithms have been proposed to solve traditional artificial intelligence problems. In this paper, in order to detect the version that can provide the best trade-off in terms of time and accuracy, convolutional networks of various depths have been implemented. Batch normalization is also considered since it acts as a regularizer and achieves the same accuracy with fewer training steps. For maximizing the yield of the complexity by diminishing, as well as minimizing the loss of accuracy, LSTM neural net layers are utilized in the process. The image sequences are proven to be classified by the LSTM in a more accelerated manner, while managing better precision. Concretely, the more complex the CNN, the higher the percentages of exactitude; in addition, but for the high-rank increase in accuracy, the time was significantly decreased, which eventually rendered the trade-off optimal. The average improvement of performance for all models regarding both datasets used amounted to 42%.

Keywords: batch normalization; convolutional neural networks; deep learning; image classification; knowledge extraction; LSTM neural networks; recommendation systems

1. Introduction

Nowadays, the quantity of available data keeps increasing on a daily basis, mainly because of the vast production that an average user can provide. This multimedia type of information has made possible new ways of processing data and understanding how to restructure them for our daily needs. However, still, we are not taking full advantage of the meta-information that the data present to us, mainly because of the lack of our ability to pinpoint their inherent visual representation. In order to fulfil this ever-growing need, the creation of systems that can explain a visual piece of information through semantic and algebraic features is deemed highly integral.

In most cases, dividing the procedure of representing and deducting the content of an image can be arranged just as in the work of [1]. In the beginning, the features that are to be produced must consist of a low-level representation, before advancing to more perplexing stages. Most common of these are color-related features, the texture of an image or even its shape. This low-level representation of characteristics can be easily approximated with a single vector representation as a flattened version of the image matrix. In order to reduce their sensitivities to potential information loss and ambiguity, it is attempted to provide them with more context-specific descriptors, since the aforementioned features are further developed. Such attempts were introduced in the works of SIFT [2], SURF [3] and HoG [4]. These newly-found descriptors appear to have a more robust behavior on homography distortions, especially when it comes to a comparison with the global features. To summarize the extracted knowledge of such procedures, the spatial features are rearranged with bag of visual words [5].

Another work heavily based on descriptor extraction was produced by [6]. The L_2 norm was utilized in this work, during the training and testing steps, mainly to create the multi-dimensional feature maps. These descriptors were easily adapted to Siamese networks with non-corresponding patches, thus enabling its utility in every algorithm pertaining to the logic of SIFT.

Studies have recently indicated that there does not exist a single procedure to characterize all datasets with the same features accordingly. However, the pure unprocessed data can often provide the most beneficial routes to extract the desired features of the available data, in contrast to the ability of a human to manually assign labels according to his/her general observations. This human-machine contradiction and comparison is also seen in the work of [7]. Despite the obvious ability of humans to correctly label faces, especially widely-known people that can rarely be misunderstood, their proposed network conglomerate of layers achieved almost identical accuracy, as well as false positive rates along with the manual human labeling. Of course, the time that the algorithm consumed to rank each face image, in comparison to the human labeling, was negligible, as the computer processing the images needed 0.33 seconds per image for the complete assignment of a label to a face.

Currently, the breakthrough of deep learning as a subfield of machine learning [8] has introduced a new idea that consists of translating the pixels of a matrix to an applicable form through iterative algorithmic learning. Most algorithms that pertain to the deep learning field pursue a high-level generalization of the available data at hand, through a hierarchical stack of processing layers. Each paralleled layer can produce a milestone output that has concluded a partial analysis on the starting image. Many state-of-the-art deep neural networks have peaked at substantial performances through an abundant layer depth analysis on the corresponding datasets. As the depth of the stack increases, so does the complexity. The two metrics are not always analogous, and in particular situations, a bigger neural network may be deemed detrimental in comparison to a smaller, more compact and, thus, more sufficient in terms of algorithmic time needed.

Taking a step further in the logical continuum of deep learning, the computational power of the current technology can be utilized to a great extent in order to detect the desired feature descriptors for input images. Deep learning algorithms in their nature are able to train themselves through the incoming input data in order to create the high level abstraction that describes the data. The most appropriate kind of architecture for such tasks is comprised by Convolutional Neural Networks (CNNs). The CNNs posses a layer stack that convolves the input image against a number of filters before they present the final result. These filters are an inextricable part of the CNN and contribute immensely to the simple convolution.

The primary contribution of this article consists of the following aspects. Initially, a framework capable of generating hierarchical labels, by integrating the powerful Convolutional Neural Networks (CNN), used to generate discriminative features, is introduced. Secondly, batch normalization is utilized so as to allow the use of much higher learning rates and be less careful about initialization. Specifically, it acts as a regularizer, whereas when applied to a state-of-the-art image classification model, batch normalization achieves the same accuracy with fewer training steps and beats the original model by a significant margin. In the following, the LSTM neural net layers, which frame the CNN schema, are properly introduced. Following the convolution feature extraction procedure, the long short-term memory NNs tackle a variation of a min-max problem; this neural layer attempts to maximize the yield of the complexity tendency to diminish, while simultaneously minimizing the loss of accuracy in the process. The input data feature maps, as generated by the CNNs, are considered a sequential streamline of images. Such image sequences can be classified by the LSTM in a more accelerated manner, as well as accomplish appealing amounts of approximately indistinguishable accuracy.

In initial efforts, such as [9], the authors used more modern CNNs for encoding, and thus, they replaced feed-forward networks with recurrent neural networks [10]. In particular, the LSTM cell has been utilized, as it outperforms other competing methods and also learns to solve complex, artificial tasks that no other recurrent net algorithm has solved [11]. The common theme of these works is that they represented images as the top layer of a large CNN (hence, the name "top-down", as no individual objects are detected) and produced models that were end-to-end trainable. Furthermore, a multiplicative input gate unit is introduced in order to protect the memory contents stored in a linear unit from perturbation by irrelevant inputs. The resulting, more complex unit is called a memory cell, where its cell is built around a central linear unit with a fixed self-connection [12]. We differentiate our proposed work from the one in [12], as optimization techniques and different metrics for validating the improvement of the proposed architecture are incorporated.

The remainder of the paper is structured as follows: Section 2 overviews related work. Section 3 presents in detail the techniques, modules and sub-modules of our model that have been chosen, while in Section 4, our proposed system architecture is presented, as well as details of the implementation and the evaluation conducted in Section 4. Ultimately, Section 5 presents conclusions and draws directions for future opus that may extend the current version and performance of this work.

2. Related Work

One of the most critical branches of machine learning consists of deep learning; a thorough procedure that attempts to classify given data through a hierarchical structuring of their meta-information. The currently proposed work is widely based on an applied domain of artificial intelligence, such as computer vision [13,14]. The main reasons behind the steep escalation of deep learning are divided into the following partitions: the current processing power of GPU technology, the ever diminishing retail cost of hardware and, of course, the breakthrough of the related machine learning science in general.

Regarding the Convolutional Neural Networks (CNN) in particular, they constitute a classification procedure where a stack of layers is fitted on a specific set of data [15]. CNNs find great use in computer vision specifically, as they have been considered the most effective approach of the related field. They can tackle issues that have been until recently deemed unsolvable, as they can rapidly provide descriptions for numerous images at once [16–18]. Flexible as they are, in depth, as well as their layer stack width, they possess the capability of refining said image descriptions and fine-tuning their statistics. Hence, CNNs in comparison to the classic neural nets are vastly more advanced and rapid, as a consequence of their lower parameter and connection complexity. Their predominant disadvantage lies in the dataset that is needed to train the model. In contradiction to the classic Bayesian NNs, where over-fitting is easily avoided, the classic datasets are susceptible to this phenomenon when the data available are not big enough.

In [19], the authors attempted to create new and more specific descriptors for the images; outlines that stem from a local proximity effect of the given training data. The combination of the locally connected descriptors provides a great reduction in the problem's dimensionality, as well as new ground truths for the available data.

An attempt to normalize CNN input has been done by [20], in order to address the issue of the internal covariate shift. During a deep neural network training, many issues emerge due to image abnormalities of the specific information distribution. Each fitted node of the layer changes with every iteration, thus significantly delaying the model learning rate, as well as its total convergence. Such input irregularities cause the model training to be exponentially harder. Therefore, in order to avoid such complications, we implement a layer of batch normalization in the proposed schema, thus reducing the issue of the internal covariate shift and rendering the need for immediate dropout layers optional.

In contrast, the authors in [21], present a Bayesian CNN, which can efficiently provide a robust algorithmic behavior when it comes to the over-fitting of smaller datasets, compared to the previously elaborated CNNs. Their main difference is the probabilistic concept that is introduced in the Bayesian CNNs, where a distribution is scattered on the kernels, and this theoretically supports the idea of inference in the Bayesian neural networks in lieu of classic dropout training.

Furthermore, works in [22], [23] propose a probabilistic network scheme, e.g. inference network. This probabilistic model consists of four component levels, which takes as input the belief of the user for each query (initially, all entities are equivalent) and produces a new ranking for the entities as output. In [22], authors propose a semantically driven Bayesian Inference Network, incorporating semantic concepts so as to improve the ranking quality of documents. Similarly, in [23], authors consider the emotions associated to brand love appearing in the form of terms in users' Twitter posts. Building on existing work that identifies seven dimensions in brand love, they propose a probabilistic network scheme that employs a topic identification method so as to identify the aspects of the brand name.

Moreover, in [24], the authors captured a compelling dataset of human movement and motion that characterized the labeled movements. Then, a framework of the deep learning field was created, which was applied on the dataset and managed to predict successfully a high percentage of queries regarding future human movements in immediate succession.

One of the major matters of contention in the deep learning subfield consists of the neural net converging point. There exists an analogous connection between the depth of the network layer and model accuracy. This degradation, while relevant to the existing amount of data, is inconsequential to overfitting, thus rendering any extra layer that may be added to the stack detrimental to the overall performance. How [25] to tackle this issue was the introduction of a framework consisting of a deep, residual learning with layers that fit a similar residual mapping, this reducing the probabilistic fitting that a specific underlying mapping could provide. More specifically, this fundamental, latent mapping $H(x)$, as well as the non-linear layers fit another aspired mapping of $F(x) := H(x) - x$, where the initial mapping is then denoted as $F(x) + x$.

Such difficulties were addressed also in the work of [26], where the authors experimented heavily with the intricacies of training a deep neural network, through understanding how and when gradient descent works more efficiently and when it does not. They experimented with non-linearities and their influence on the model, as well as how the activation functions were initialized. Their results concluded in a new initialization scheme to overcome the aforementioned issues.

In the work of [27], the corresponding authors made a proposal for extending a classic activation function of deep learning, the ReLU. The main concept of this extension was called Parametric Rectified Linear Unit (PReLU), and its main activity pertained to the adaptation of the model on learning the rectifiers' hyper-parameters, as well as maintaining the computational complexity, while boosting the accuracy.

One of the biggest contribution of the field, as elaborated in the work of [14], is the ImageNet convolutional neural net, which was used in the ILSVRC (ImageNet Large Scale Visual Recognition Competition) competitions (http://www.imagenet.org/) of 2010 as well as 2012 and was trained with the marginally highest results. The main contribution of this authors' work was the paralleled optimization of the 2D convolution on GPU units, as well as the rest of the operations taking place in the CNNs. The newly-found features of this proposal increased the model's performance, but also the time consumed for the execution of the algorithm.

Concerning the recurrent layers of the schema, [28] et al. in their work implemented an empirical evaluation and comparison of different RNNs (Recurrent Neural Networks) such as the Gated Recurrent Units (GRUs) and the Long Short-Term Memory Units (LSTM). During their experiments on speech signal modeling, it was evidently revealed that *tanh* units were mainly out-scaled in terms of performance by the advanced GRU units. Finally, the authors in [29] analyzed a Convolutional Long Short-Term Memory recurrent Neural Network (CNN-LSTM) aiming to successfully recognize gestures of varying duration and complexity.

3. Material and Methods

In this section, a formal definition of all the layers that constitute our hybrid schema and a brief description of the semantics that lie behind each case are presented.

3.1. Convolutional Layer

The two main parts that constitute a convolutional neural net that is applied as a layer to the input data are the following. The sequence of the input-stream is distributed to a number of $d^{'}$ filters, whose size amounts to r, $F \in \Re^{d^{'} \times r}$:

$$f_t = \phi(F[e_{t-(r/2)+1}; \ldots ; e_t; \ldots ; e_{t+(r/2)}]) \tag{1}$$

The activation function of our layer is denoted as ϕ. Other options, such as tanh or a rectifier unit, exist, although a non-linear function is chosen. For every part of the sequence of inputs that the function is applied to, a part of the resulting sequence is produced that we denote as $F = (f_1, f_2, \ldots, f_T)$.

The product of the convolutional layer F is then followed by a max-pool layer with a kernel of according to size $r^{'}$, while all the elements of the resulting vectors are processed through max, thus producing a scalar product:

$$f_t^{'} = max(f_{(t-1)\times r^{'}+1}, \ldots, f_{t\times r^{'}}) \tag{2}$$

$$F^{'} = (f_1^{'}, f_2^{'}, \ldots, f_{T/r^{'}}^{'}) \tag{3}$$

3.2. ReLU Nonlinearity

The standard way to model a neuron's output f as a function of its input x is with $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = max(0, x)$. Following the work presented in [30], we refer to neurons with this non-linearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with *tanh* units. Rectified linear units, compared to the sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets.

3.3. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) constitutes an effective way of training deep networks, similar to the ones utilized in the present manuscript. Concretely, SGD variants, like momentum Adagrad [31,32], have been used in other studies in order to achieve state-of-the-art performance. What is more, SGD optimizes the parameters Θ of a concrete network so as to minimize the loss, as mentioned in the following equation:

$$\Theta = argmin \frac{1}{N} \sum_{i=1}^{N} l(x_i, \Theta) \tag{4}$$

where $x_{1,\ldots,N}$ is the training dataset. With the use of SGD, the training phase is utilized in steps, whereas at each step, a mini batch $x_{1,\ldots,m}$ of size m is considered. Regarding mini-batch, it is commonly used in order to approximate the gradient of the loss function with respect to the parameters. Therefore, the following computation is considered:

$$\frac{1}{m} \frac{\partial l(x_i, \Theta)}{\partial \Theta} \tag{5}$$

The advantages of using mini-batches of several examples instead of using mini-batches of one example are considered as follows. Initially, the gradient of the loss over a mini-batch is regarded an estimate of the gradient over the training set, where its quality improves as the batch size increases. In the following, the parallelism afforded by the modern computing platforms drives the much more efficient computation over a batch than the m computations for different individual examples.

On the other hand, despite the fact that stochastic gradient descent is simple and effective, it requires careful tuning of the model hyper-parameters. These parameters include the learning rate used in optimization, along with the initial values regarding the model parameters. More specifically, the training phase is overly complicated since the inputs to each layer are affected by the parameters of all preceding layers; as a result, small changes in the network parameters are amplified while the network becomes deeper.

3.4. Batch Normalization Layer

Furthermore, the nature of each layer's parameters implies a great complication as far as the convergence of the model is concerned. A respectable amount of deviations is observed, on the values of the input layers with each passing feature set. Therefore, there emerges a significant need to normalize each feature, within a given amount of previous and subsequent feature vectors, which are denoted as batches. Thus, a multi-dimensional input, which is accepted by a layer, $x = (x^{(1)} \dots x^{(d)})$, has each of the d dimensions normalized in terms of mean and variance:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{6}$$

Following the application of the aforementioned procedure on the input layers, the newly-normalized values of the vectors are denoted as, $\hat{x}_{1,\dots,m}$, while the linear transformations of $\hat{x}$ are denoted as $y_{1,\dots,m}$.

Therefore, the complete layer of the batch normalization is completed as presented:

$$BN_{\gamma,\beta} \colon x_{1,\dots,m} \rightarrow y_{1,\dots,m} \tag{7}$$

The BN transform, as the batch normalizing procedure is denoted, is further elaborated with its according functions in Table 1. The constant ϵ, which is utilized in the corresponding functions, contributes to the numerical stability of the problem.

Table 1. Applying the batch normalizing layer on a given batch of inputs.

Description	**Function**
Input	Values of x over a batch: $B = \{x_{1,\dots,m}\}$;
Input: Parameters to be learned	γ, β
Output	$\{y_i = BN_{\gamma,\beta}(x_i)\}$
Mini-batch mean	$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
Mini-batch variance	$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
Normalize	$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
Scale and shift	$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

3.5. LSTM Layer

The LSTM neural network can be coherently presented as a set of sequential functions that process the data input layer, in between the current and the subsequent hidden states of the algorithm. The final product p_{t+1} of the LSTM neural net whose input is denoted as x_t, m_t and c_t the aforementioned sequence of hidden states, is produced through the following equations:

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1}) \tag{8}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \tag{9}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1}) \tag{10}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{cx}x_t + W_{cm}m_{t-1}) \tag{11}$$

$$m_t = o_t \odot c_t \tag{12}$$

$$p_{t+1} = Softmax(m_t) \tag{13}$$

What f_t denotes is the "forget gates" of the network. The model's "forget gates" have the capability of ignoring specific past states of certain memory cells. On the other hand, the input gates i_t of the model have the power to ignore certain parts of the input in contrast to the previous cell states. The output gate is denoted by o_t, whose purpose is to administer a filter to the current memory cell. This is then taken to the final hidden state.

These gates and their conglomerate structure allow this version of recurrent neural networks to map dependencies between specific inputs and the long-term behavior of the network, as well as to avoid gradients whose value is exponentially surging to inconvenient peaks.

3.6. Dropout

The dropout layer [33], which is applied sequentially following the previous stack of filters, acts as a regularizing tool of the input data when deep neural networks are concerned. Let the output product of a neural network, which is L layers deep, be denoted as $\widehat{y}$, initialized with an according loss function $E(\cdot\,,\cdot\,)$ with the softmax loss function as a potential loss function or the Euclidean loss function, which is also often called square loss. Then, all weights of each dimension of the neural network are initialized, through a corresponding matrix, which will contain W_i, $K_i \times K_{i-1}$, as well as b_i, which denotes the bias, is applied with each parsing of the network, where the dimensions are K_i and each layer $i = 1, \ldots, L$.

The optimization of the neural network often requires another variable that acts as a regularizer of the input. This variable is frequently the L_2 norm, reduced by a certain λ decay rate, before achieving its latter goal of minimization [21,34]:

$$L_{dropout} := \frac{1}{N}\sum_{i=1}^{N} E(y_i, \widehat{y}_i) + \lambda \sum_{i=1}^{L}(\|W_i\|_2^2 + \|b_i\|_2^2) \tag{14}$$

What a dropout makes possible is the sampling of the binary variables of the network, for every input and layer accordingly. These variables are initialized with a certain value of one. The probability for their initialization is p_i, where i denotes the specific current layer. If this value is not one, but is initialized with zero, then it is decided that this specific unit is dropped. This procedure and its variables are then utilized during the backward pass propagation of the model's parameters.

The application of the dropout layer often takes place following the last convolutional layer of a given network, as well as after a recurrent type of layer, such as LSTMs or GRUs, if existent. Below, the inputs x_t of the recurrent layers are presented, before the application of said dropout layer, where $f_t^{'}$ denotes the output, which the last convolutional layer of the stack has produced:

$$x_t = f_t^{'} \tag{15}$$

With the addition of the dropout technique, the following are presented:

$$r_t^i \sim Bernoulli(p) \tag{16}$$

$$x_t = r_t \odot f_t^{'} \tag{17}$$

The variable p denotes the dropout probability. This is often set to 0.5; thus, Equation (16) notes that the dropout vectors are produced through the binary nature of the probability density function of Bernoulli. As for r_t^i, this denotes the i-th element of the binary $r_t \in \Re^{d^{'}}$ vectors.

3.7. Input Representation

As has been introduced in the literature of recurrent neural networks, the information contained in the sequence of words $S_0, S_1, \dots, S_t$ at a certain time $t+1$ is represented by a fixed length hidden state h_t. The following hidden state constitutes a non-linear function of the past hidden state, as well as the current input, which produces an updated memory that can capture, through time, non-linear dependencies:

$$h_{t+1} = f(h_t, x_{t+1}) \tag{18}$$

Regarding the non-linear function f, it is of utmost importance to efficiently represent the corresponding inputs x of our model. Specifically, the non-linear function that is implemented in the present manuscript is the LSTM one. As mentioned above, the LSTM cell [11] has increasingly become popular in recent years due to the fact that it has the potential to capture long-term dependencies in sequence prediction problems. Moreover, LSTM has the ability to cope with the vanishing/exploding gradient problems in recurrent neural networks.

As it is already known, deep convolutional neural networks have achieved state-of-the-art performances regarding the field of image classification in recent years. Therefore, the use of a convolutional neural network for mapping images I to fixed length vector representations is considered in order to represent images strategically. Specifically, the architecture of GoogleNet [35], which employs an innovative batch normalization technique, is used. In the following, $x = CNN(I)$ is considered as a $D_i \times 1$ vector, where D_i is the fixed dimension of any input that is given to the LSTM.

4. Experiments and Results

4.1. Dataset Description

The dataset used in the experiments was the MNIST database (http://yann.lecun.com/exdb/mnist/) [15]. MNIST consists of a collection of 60,000 hand written digits in image-oriented matrix format. Each image is associated with a specific label, which signifies the expected digit prediction according to the digit intention of the writer. This labeled dataset can be easily adapted to classification algorithms; regarding the current work, the labeled images iteratively trained our proposed neural network. Additionally, our model was also trained on another dataset, CIFAR10, in order to further evaluate the contribution of our work. CIFAR10, just like MNIST, is a collection of images, correspondingly labeled with a number to represent ten different classes.

4.2. Implementation

As presented in Figure 1 and Table 2, the flow of information starts with the layer of the input image. The abbreviations for the different methods utilized in the present manuscript are introduced in Table 3. As provided by the MNIST dataset, all images of the experiments were of a rectangular shape and of 28×28 dimensions exactly. In the case of the CIFAR10 dataset, the initial dimensions were 32×32. Each image of the corresponding batch was then proceeded in a series of convolutional layers where the first part consisted of a 32-channel decomposition of the image. The initial kernel size was 5×5, and although the dimensions of the image remained the same, the number of channels constituted units that were easily adapted to the ReLU (Rectified Linear Units) activation function. The following layer consisted of a max pooling procedure where all 32 channels were directly fed to a sample-based discretization process of max pooling. There, the dimensionality of the input representation was reduced, according to the decided kernel size. Since the pooling's kernel was 2×2, our new dimensions for the channels, known as feature maps, were 14×14 (16×16 for CIFAR10 dataset). To conclude the initial set of layers, the batch normalization technique was performed. Concretely, the gradients of the batch were less vulnerable to outliers, and a normalized range was created within the mini-batch, thus accelerating the learning process by allowing a faster convergence for the model. Please note that the aforementioned process was repeated two and three times for the two different proposed models, respectively.

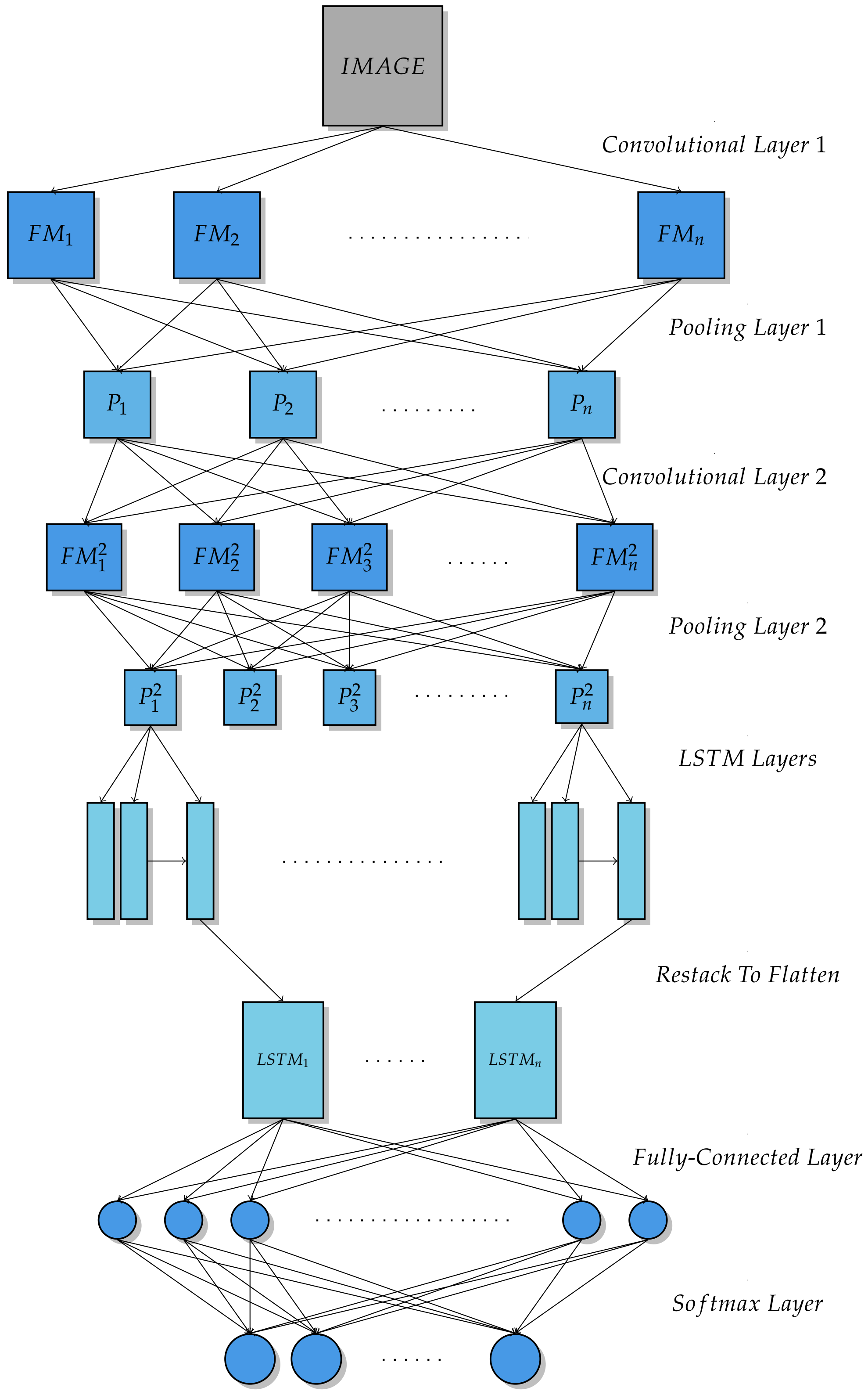


Figure 1. Network infrastructure (where FM_i is the i-th feature map and P_i is the i-th pooling layer).

Thus, the first step of our proposed schema consisted of this particular set of layers and could be replicated in matters of iterations and variable initialization. The second step of our process consisted of the restructuring of the LSTM units' layer in order to accelerate the process accordingly. As elaborated in the Introduction, in our proposed schema, a number of LSTM units was interposed between the convolutional and the dense layers. A separate LSTM layer was assigned for each feature map produced by the convolutional layers. Their output was directly relevant to the selected number of output units, which was initialized as 1 in our schema, and was constructed together through a recompositioning step, arranging all output vectors as a 2-dimensional single output. Before completing the schema with the dense and logit layers, the constructed LSTM output was flattened in a single long vector, in order to adapt to the inputs of the conclusive fully-connected layer and produce the probabilities of the ten-digit classifier.

Table 2. Abbreviations for different methods utilized.

Layer	Dimensions	Channels	Kernel Size
Input Image	28×28	1	
Convolutional/ReLU	28×28	32	5×5
Max Pooling Layer	14×14	32	2×2
Batch Normalization	14×14	32	
Convolutional/ReLU	14×14	64	3×3
Max Pooling Layer	7×7	64	2×2
Batch Normalization	7×7	64	
LSTM Layer	7×1	64	
Fully-Connected/Dense		1000	
Dropout Layer			
Fully-Connected/Softmax		10	

4.3. Evaluation

Regarding the results as presented in Tables 4–7, as well as Figures 2 and 3, a safe assumption can be extracted concerning the fluctuations in accuracy of the concerned methods. More specifically, the performance of the proposed schemes in terms of accuracy, loss (cross-entropy), in accordance with execution time (in seconds) has been measured. All results are comprised of the average out of five different runs in order to avoid outliers.

As expected, the baseline approach that contained a regular convolutional scheme maintained a steady and linear progression in its accuracy, as well as in the time needed for the completion of the experiments. In the event of including a batch normalization layer after the convolutional net, the total completion time of the corresponding experiments marginally increased. That chronic increment amounts to the sum of computations applied during the reduction of the covariate shift.

Making the convolutional network deeper, thus more complex in dimensionality, will affect the model's performance. In the present work, convolutional networks of various depths were implemented in order to detect the version that could provide the better time-accuracy trade-off. Indeed, the more complex CNN managed to achieve higher percentages of accuracy. Despite the time needed, accuracy was increased in high ranks, which made the trade-off suboptimal.

The conglomerate structure of the previous baseline network, in sequence with our additional LSTM neural net logic, yielded results within our expected range of values. The precision of our novel approach scaled increasingly as the model was trained with additional epochs. In comparison with the baseline methods, similar results were achieved with a standard deviation of 2×10^{-3}, while also occasionally yielding surpassing accuracy scores. However, on the contrary, in terms of the experiment completion time, the results highlighted our approach as importantly more rapid in execution. While maintaining similar amounts of accuracy, our novel CNN-LSTM technique managed to reduce the execution time by 30% during the first unstable epochs and by approximately 42% after convergence.

More specifically, after 50 epochs of training on the MNIST dataset, our model achieved a 36% and 44% improvement on its results on the 2-CNN model and 3-CNN model, respectively; that is, regarding the comparison of the model that contained our contribution with the CNN-LSTM architecture against the model that achieved the highest accuracy. With the gradual increase of the data that were supplied for the training step of the models, their percentages were scaled to 42% and 40%, again, respectively. Regarding the experiments on the CIFAR10 database, our model provided similar results in terms of accuracy and execution time, as expected. The first results of the execution time comparison presented a 42% improvement on both the 2-CNN, as well as 3-CNN layer models. Providing a bigger amount of data, just like the previous experiment with MNIST, yielded an improvement of 41% and 43% accordingly.

Table 3. Abbreviations for different methods utilized.

Abbreviation	Description
C1	First Convolutional Layer
C2	Second Convolutional Layer
C3	Third Convolutional Layer
B_N	Batch Normalization
LSTM	Long Short-Term Memory Neural Layer

Table 4. Accuracy, loss and time for different numbers of epochs (10% sample) for the MNIST dataset.

Methods	Accuracy (Percentage)	Loss (Cross-Entropy)	Time (s)
Epoch = 1			
C1 + C2	87.7	0.43	104
C1 + C2 + B_N	92.8	0.2	107
C1 + C2 + B_N + LSTM	90.8	0.314	78
C1 + C2 + C3	87.7	0.38	149
C1 + C2 + C3 + B_N	95	0.16	149
C1 + C2 + C3 + B_N + LSTM	87.1	0.41	111
Epoch = 5			
C1 + C2	94.3	0.15	448
C1 + C2 + B_N	97.1	0.097	519
C1 + C2 + B_N + LSTM	95.8	0.137	278
C1 + C2 + C3	95.6	0.145	595
C1 + C2 + C3 + B_N	97.2	0.084	768
C1 + C2 + C3 + B_N + LSTM	96	0.11	467
Epoch = 10			
C1 + C2	95.3	0.15	956
C1 + C2 + B_N	97.1	0.097	1048
C1 + C2 + B_N + LSTM	95.8	0.137	756
C1 + C2 + C3	96.2	0.13	1262
C1 + C2 + C3 + B_N	97.5	0.095	1356
C1 + C2 + C3 + B_N + LSTM	97.3	0.09	810
Epoch = 20			
C1 + C2	96.1	0.101	1961
C1 + C2 + B_N	97.2	0.085	2174
C1 + C2 + B_N + LSTM	97.3	0.081	1378
C1 + C2 + C3	97	0.12	2510
C1 + C2 + C3 + B_N	97.6	0.092	2741
C1 + C2 + C3 + B_N + LSTM	97.4	0.09	1696
Epoch = 40			
C1 + C2	97	0.101	3866
C1 + C2 + B_N	97.7	0.098	3887
C1 + C2 + B_N + LSTM	97.5	0.088	2484
C1 + C2 + C3	97.3	0.13	5080
C1 + C2 + C3 + B_N	97.8	0.097	5395
C1 + C2 + C3 + B_N + LSTM	97.6	0.09	3007

Table 4. *Cont.*

Methods	Accuracy (Percentage)	Loss (Cross-Entropy)	Time (s)
	Epoch = 50		
C1 + C2	96.9	0.102	4790
C1 + C2 + B_N	97.7	0.097	4794
C1 + C2 + B_N + LSTM	97.6	0.082	3045
C1 + C2 + C3	97.4	0.12	6351
C1 + C2 + C3 + B_N	97.8	0.097	6809
C1 + C2 + C3 + B_N + LSTM	97.8	0.091	3812

Table 5. Accuracy, loss and time for different numbers of epochs (20% sample) for the MNIST dataset.

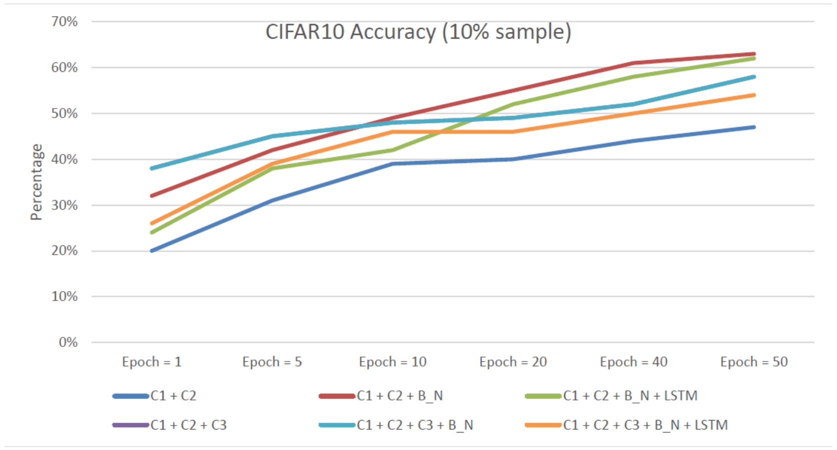
Methods	Accuracy (Percentage)	Loss (Cross-Entropy)	Time (s)
	Epoch = 1		
C1 + C2	91	0.28	185
C1 + C2 + B_N	95.9	0.123	186
C1 + C2 + B_N + LSTM	93.9	0.205	113
C1 + C2 + C3	94	0.208	228
C1 + C2 + C3 + B_N	96.5	0.12	256
C1 + C2 + C3 + B_N + LSTM	95.4	0.15	167
	Epoch = 5		
C1 + C2	96.5	0.086	902
C1 + C2 + B_N	97.5	0.07	903
C1 + C2 + B_N + LSTM	97.4	0.081	640
C1 + C2 + C3	96	0.106	982
C1 + C2 + C3 + B_N	97.7	0.068	1027
C1 + C2 + C3 + B_N + LSTM	97.4	0.076	677
	Epoch = 10		
C1 + C2	97.4	0.083	1805
C1 + C2 + B_N	97.5	0.067	1814
C1 + C2 + B_N + LSTM	97.7	0.076	1222
C1 + C2 + C3	97.5	0.084	2001
C1 + C2 + C3 + B_N	98	0.061	2076
C1 + C2 + C3 + B_N + LSTM	97.9	0.068	1272
	Epoch = 20		
C1 + C2	97.9	0.069	3635
C1 + C2 + B_N	98.3	0.063	3708
C1 + C2 + B_N + LSTM	97.7	0.075	2354
C1 + C2 + C3	97.7	0.083	4017
C1 + C2 + C3 + B_N	98.1	0.061	4152
C1 + C2 + C3 + B_N + LSTM	98	0.066	2576
	Epoch = 40		
C1 + C2	98.1	0.076	7572
C1 + C2 + B_N	98.2	0.059	7657
C1 + C2 + B_N + LSTM	97.8	0.074	4688
C1 + C2 + C3	97.9	0.098	8103
C1 + C2 + C3 + B_N	98.3	0.068	8296
C1 + C2 + C3 + B_N + LSTM	98	0.065	5152
	Epoch = 50		
C1 + C2	98.1	0.075	9477
C1 + C2 + B_N	98.2	0.059	10,058
C1 + C2 + B_N + LSTM	98	0.075	5811
C1 + C2 + C3	97.9	0.098	10,113
C1 + C2 + C3 + B_N	98.3	0.07	10,384
C1 + C2 + C3 + B_N + LSTM	98.1	0.066	6312

Table 6. Accuracy, loss and time for different numbers of epochs (10% sample) for the CIFAR10dataset.

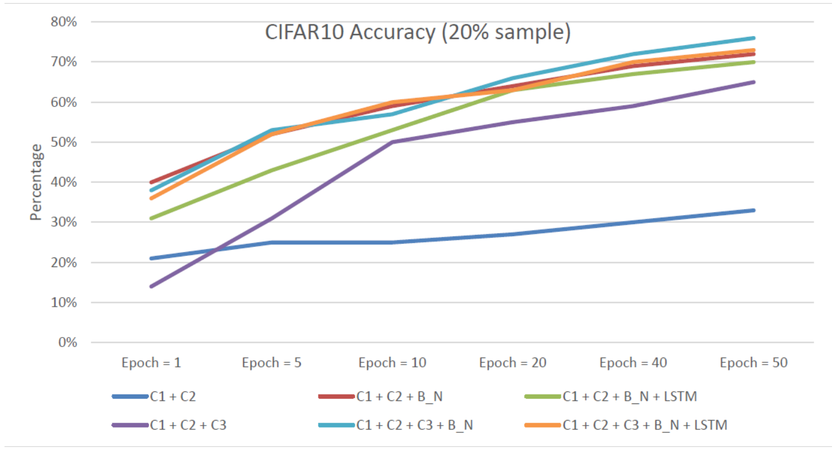
Methods	Accuracy (Percentage)	Loss (Cross-Entropy)	Time (s)
	Epoch = 1		
C1 + C2	20	2.3	160
C1 + C2 + B_N	32	2.43	121
C1 + C2 + B_N + LSTM	24	2.49	70
C1 + C2 + C3	23	2.21	150
C1 + C2 + C3 + B_N	38	2.11	163
C1 + C2 + C3 + B_N + LSTM	26	2.08	80
	Epoch = 5		
C1 + C2	31	2.28	720
C1 + C2 + B_N	42	1.52	553
C1 + C2 + B_N + LSTM	38	1.83	357
C1 + C2 + C3	31	1.82	726
C1 + C2 + C3 + B_N	45	1.51	754
C1 + C2 + C3 + B_N + LSTM	39	1.59	371
	Epoch = 10		
C1 + C2	39	2.29	1239
C1 + C2 + B_N	49	1.7	1171
C1 + C2 + B_N + LSTM	42	1.71	562
C1 + C2 + C3	43	1.82	1451
C1 + C2 + C3 + B_N	48	1.71	1432
C1 + C2 + C3 + B_N + LSTM	46	1.73	820
	Epoch = 20		
C1 + C2	40	2.29	2240
C1 + C2 + B_N	55	2.01	2302
C1 + C2 + B_N + LSTM	52	2.1	1213
C1 + C2 + C3	44	1.81	2581
C1 + C2 + C3 + B_N	49	2.03	2766
C1 + C2 + C3 + B_N + LSTM	46	2.03	1687
	Epoch = 40		
C1 + C2	44	2.27	4476
C1 + C2 + B_N	61	1.81	4595
C1 + C2 + B_N + LSTM	58	1.96	2528
C1 + C2 + C3	47	1.85	5171
C1 + C2 + C3 + B_N	52	1.94	5662
C1 + C2 + C3 + B_N + LSTM	50	1.89	3259
	Epoch = 50		
C1 + C2	47	2.26	5712
C1 + C2 + B_N	63	1.84	5772
C1 + C2 + B_N + LSTM	62	1.91	3093
C1 + C2 + C3	51	1.82	6662
C1 + C2 + C3 + B_N	58	1.87	7107
C1 + C2 + C3 + B_N + LSTM	54	1.79	4065

Table 7. Accuracy, loss and time for different numbers of epochs (20% sample) for the CIFAR10 dataset.

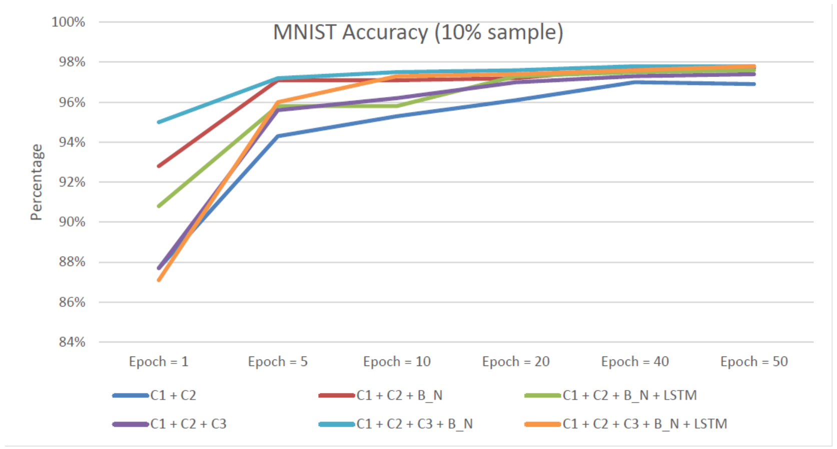
Methods	Accuracy (Percentage)	Loss (Cross-Entropy)	Time (s)
	Epoch = 1		
C1 + C2	21	2.29	211
C1 + C2 + B_N	40	1.85	201
C1 + C2 + B_N + LSTM	31	1.81	122
C1 + C2 + C3	14	2.27	257
C1 + C2 + C3 + B_N	38	1.91	225
C1 + C2 + C3 + B_N + LSTM	36	1.85	132
	Epoch = 5		
C1 + C2	25	2.28	1037
C1 + C2 + B_N	52	1.54	1051
C1 + C2 + B_N + LSTM	43	1.53	582
C1 + C2 + C3	31	2.29	1291
C1 + C2 + C3 + B_N	53	1.37	1311
C1 + C2 + C3 + B_N + LSTM	52	1.36	652
	Epoch = 10		
C1 + C2	25	2.28	2035
C1 + C2 + B_N	59	1.64	2152
C1 + C2 + B_N + LSTM	53	1.44	1205
C1 + C2 + C3	50	2.18	2415
C1 + C2 + C3 + B_N	57	1.9	2407
C1 + C2 + C3 + B_N + LSTM	60	1.24	1335
	Epoch = 20		
C1 + C2	27	2.27	4107
C1 + C2 + B_N	64	1.65	4318
C1 + C2 + B_N + LSTM	63	1.4	2594
C1 + C2 + C3	55	2.15	5003
C1 + C2 + C3 + B_N	66	1.81	4995
C1 + C2 + C3 + B_N + LSTM	63	1.22	2809
	Epoch = 40		
C1 + C2	30	2.19	8265
C1 + C2 + B_N	69	1.53	8719
C1 + C2 + B_N + LSTM	67	1.38	5187
C1 + C2 + C3	59	2.11	10,089
C1 + C2 + C3 + B_N	72	1.51	10,105
C1 + C2 + C3 + B_N + LSTM	70	1.19	5817
	Epoch = 50		
C1 + C2	33	2.19	10,320
C1 + C2 + B_N	72	1.5	10,952
C1 + C2 + B_N + LSTM	70	1.33	6404
C1 + C2 + C3	65	2.09	12,631
C1 + C2 + C3 + B_N	76	1.38	12,708
C1 + C2 + C3 + B_N + LSTM	73	1.07	7158



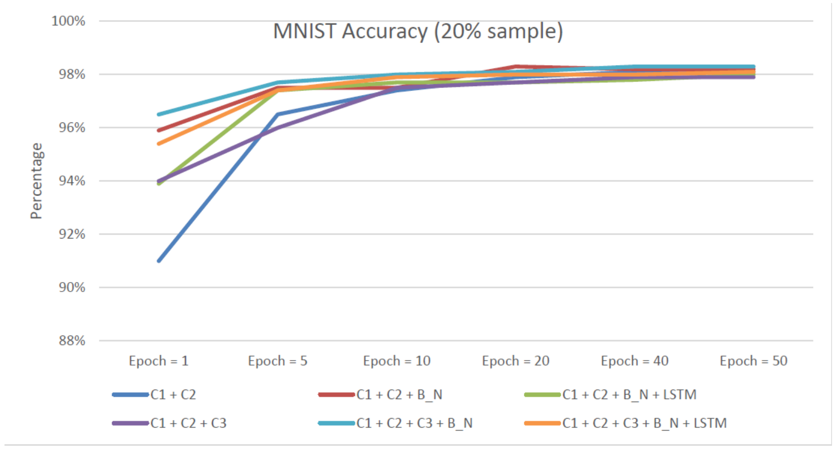
(**a**) CIFAR10 (10% sample)



(**b**) CIFAR10 (20% sample)

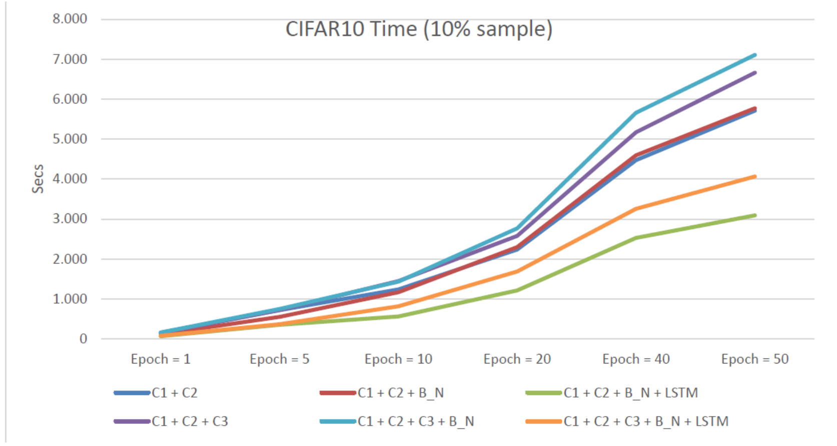


(**c**) MNIST (10% sample)

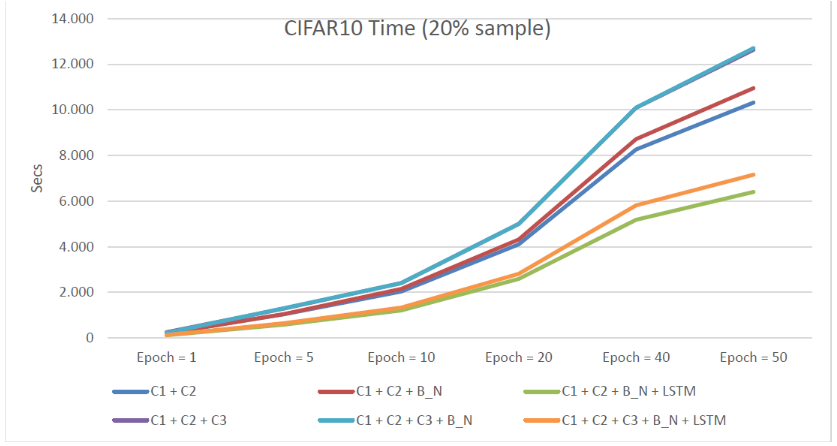


(**d**) MNIST (20% sample)

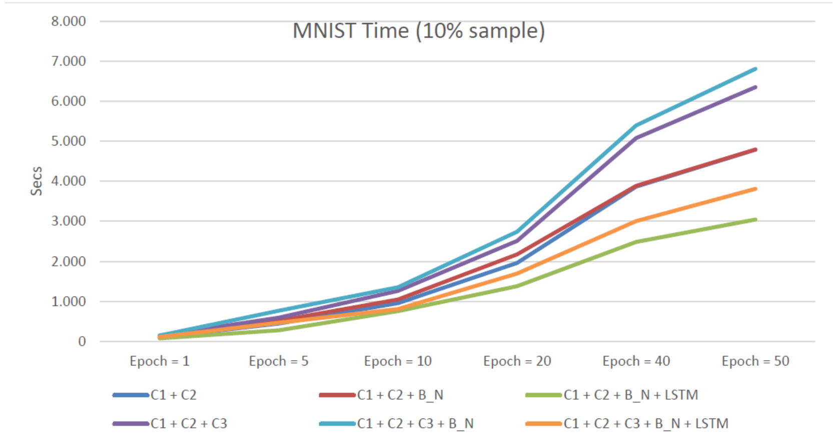
Figure 2. Accuracy (percentage) for different numbers of epochs for CIFAR10 and MNIST.



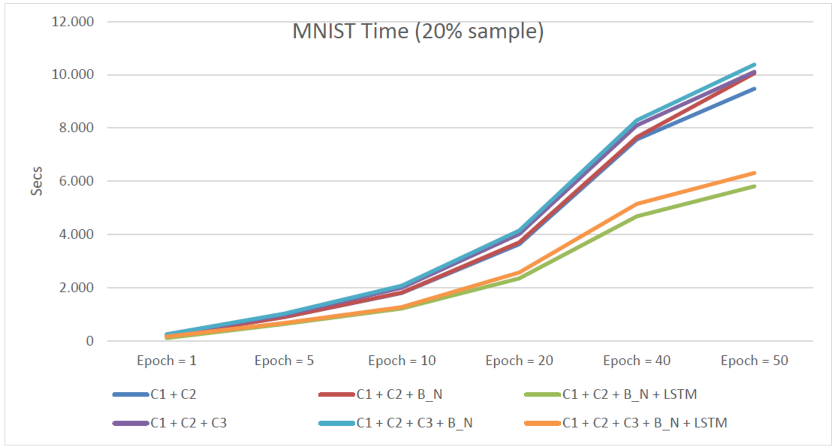
(**a**) CIFAR10 (10% sample)



(**b**) CIFAR10 (20% sample)



(**c**) MNIST (10% sample)



(**d**) MNIST (20% sample)

Figure 3. Time (s) for different numbers of epochs for CIFAR10 and MNIST.

5. Conclusions and Future Work

In the proposed work, a methodology utilizing convolutional networks as well as batch normalization and LSTM neural net layers for achieving the best trade-off in terms of time and accuracy regarding the image classification problem, have been presented. In addition, Rectified Linear Units (ReLUs) nonlinearity is introduced for faster training of deep convolutional neural networks than their equivalents with the sigmoid function or similar activation functions on large and complex datasets. On the other hand, dropout is an effective way, aiming at regularizing deep neural networks; with this process, binary variables for every input point and for every network unit in each layer are sampled.

The baseline approach, which contained a regular convolutional scheme, maintained a steady and linear progression in its accuracy, as well as in the time needed for completion of the experiments, whereas after the import of the batch normalization layer, the overall time of the corresponding experiments marginally increased. As the convolutional network deepens, thus becoming more complex in dimensionality, the model's performance becomes affected. The more complex the CNN, the higher the percentages of accuracy it managed to achieve, and despite the time needed, accuracy was increased to high ranks, which made the trade-off suboptimal. Furthermore, the accuracy of our novel approach scaled increasingly as the model was trained with additional epochs. While maintaining similar amounts of accuracy, our novel CNN-LSTM technique was successful at reducing the execution time by 30% during the first unstable epochs and by approximately 42% after convergence.

As future work, we plan to design more comprehensive CNN models as these models have achieved significant success in various computer vision tasks, including image classification, object detection, image retrieval and image captioning. It is our thought that artificial intelligence should be capable of tackling a broad set of computer vision problems. Therefore, a future plan deals with the revisiting of the vast amount of hyper parameters that such deep architectures present. The fine-tuning of the proposed model could highlight a new set of latent patterns whose appearance could not be previously detected. Furthermore, regarding the theoretical background of the work, a potential approach could be implemented concerning the complexity of the architecture. As a model deepens in terms of layers, and simultaneously in the size of its graph, there emerge new ways for defining the optimal connection within this stack of layers, and algorithmic schemes are still to be introduced.

Author Contributions: A.S., A.K., P.M. and S.S. conceived of the idea, designed and performed the experiments, analyzed the results, drafted the initial manuscript and revised the final manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guo, Y.; Liu, Y.; Oerlemans, A.; Lao, S.; Wu, S.; Lew, M.S. Deep Learning for Visual Understanding: A Review. *Neurocomputing* **2016**, *187*, 27–48. [CrossRef]
2. Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vis.* **2004**, *60*, 91–110. [CrossRef]
3. Bay, H.; Tuytelaars, T.; Gool, L.J.V. SURF: Speeded Up Robust Features. In Proceedings of the European Conference on Computer Vision (ECCV), Graz, Austria, 7–13 May 2006; pp. 404–417.
4. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), San Diego, CA, USA, 20–25 June 2005; pp. 886–893.
5. Sivic, J.; Zisserman, A. Video Google: A Text Retrieval Approach to Object Matching in Videos. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Nice, France, 13–16 October 2003; pp. 1470–1477.

6. Simo-Serra, E.; Trulls, E.; Ferraz, L.; Kokkinos, I.; Fua, P.; Moreno-Noguer, F. Discriminative Learning of Deep Convolutional Feature Point Descriptors. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 118–126.
7. Taigman, Y.; Yang, M.; Ranzato, M.; Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014; pp. 1701–1708.
8. Hinton, G.E.; Osindero, S.; Teh, Y.W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef] [PubMed]
9. Donahue, J.; Hendricks, L.A.; Guadarrama, S.; Rohrbach, M.; Venugopalan, S.; Darrell, T.; Saenko, K. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 2625–2634.
10. Kiros, R.; Salakhutdinov, R.; Zemel, R.S. Multimodal Neural Language Models. In Proceedings of the 31th International Conference on Machine Learning (ICML), Beijing, China, 21–26 June 2014; pp. 595–603.
11. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
12. Sainath, T.N.; Vinyals, O.; Senior, A.W.; Sak, H. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, Australia, 19–24 April 2015; pp. 4580–4584.
13. Ciresan, D.C.; Meier, U.; Schmidhuber, J. Multi-Column Deep Neural Networks for Image Classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June2012; pp. 3642–3649.
14. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS), Lake Tahoe Nevada, 3–8 December 2012; pp. 1106–1114.
15. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
16. LeCun, Y.; Huang, F.J.; Bottou, L. Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Washington, DC, USA, 27 June–2 July 2004; pp. 97–104.
17. Lee, H.; Grosse, R.B.; Ranganath, R.; Ng, A.Y. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML), Montreal, QC, Canada, 14–18 June 2009; pp. 609–616.
18. Turaga, S.C.; Murray, J.F.; Jain, V.; Roth, F.; Helmstaedter, M.; Briggman, K.L.; Denk, W.; Seung, H.S. Convolutional Networks Can Learn to Generate Affinity Graphs for Image Segmentation. *Neural Comput.* **2010**, *22*, 511–538. [CrossRef] [PubMed]
19. Brown, M.A.; Hua, G.; Winder, S.A.J. Discriminative Learning of Local Image Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 43–57. [CrossRef] [PubMed]
20. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* **2015**, arXiv:1502.03167.
21. Gal, Y.; Ghahramani, Z. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. *arXiv* **2015**, arXiv:1506.02158.
22. Panagiotis, G.; Andreas, K.; Christos, M.; Georgios, P. Review-based Entity-ranking Refinement. In Proceedings of the 11th International Conference on Web Information Systems and Technologies (WEBIST), Lisbon, Portugal, 20–22 May 2015; pp. 402–410.
23. Andreas, K.; Eleanna, K.; Christos, Makris. Can We Rank Emotions? A Brand Love Ranking System for Emotional Terms. In Proceedings of the IEEE International Congress on Big Data, New York, NY, USA, 27 June–2 July 2015; pp. 71–78.
24. Bütepage, J.; Black, M.J.; Kragic, D.; Kjellström, H. Deep Representation Learning for Human Motion Prediction and Classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 1591–1599.

25. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
26. Glorot, X.; Bengio, Y. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Sardinia, Italy, 13–15 May 2010; pp. 249–256.
27. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
28. Chung, J.; Gülçehre, Ç.; Cho, K.; Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv* **2014**, arXiv:1412.3555.
29. Tsironi, E.; Barros, P.V.A.; Weber, C.; Wermter, S. An Analysis of Convolutional Long Short-Term Memory Recurrent Neural Networks for Gesture Recognition. *Neurocomputing* **2017**, *268*, 76–86. [CrossRef]
30. Nair, V.; Hinton, G.E. Rectified Linear Units Improve Restricted Boltzmann Machines. In Proceedings of the 27th International Conference on Machine Learning (ICML), Haifa, Israel, 21–24 June 2010; pp. 807–814.
31. Duchi, J.C.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
32. Sutskever, I.; Martens, J.; Dahl, G.E.; Hinton, G.E. On the Importance of Initialization and Momentum in Deep Learning. In Proceedings of the International Conference on Machine Learning (ICML), Atlanta, GA, USA, 16–21 June 2013; pp. 1139–1147.
33. Srivastava, N.; Hinton, G.E.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
34. Gal, Y.; Ghahramani, Z. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 1019–1027.
35. Vinyals, O.; Toshev, A.; Bengio, S.; Erhan, D. Show and Tell: A Neural Image Caption Generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 3156–3164.