Convolutional Recurrent Neural Networks: Learning Spatial Dependencies for Image Representation

Zhen Zuo[1], Bing Shuai[1], Gang Wang[1,2], Xiao Liu[1], Xingxing Wang[1], Bing Wang[1], Yushi Chen[3]

[1]Nanyang Technological University, Singapore
[2]Advanced Digital Sciences Center, Singapore
[3]Harbin Institute of Technology, China

Abstract

In existing convolutional neural networks (CNNs), both convolution and pooling are locally performed for image regions separately, no contextual dependencies between different image regions have been taken into consideration. Such dependencies represent useful spatial structure information in images. Whereas recurrent neural networks (RNNs) are designed for learning contextual dependencies among sequential data by using the recurrent (feedback) connections. In this work, we propose the convolutional recurrent neural network (C-RNN), which learns the spatial dependencies between image regions to enhance the discriminative power of image representation. The C-RNN is trained in an end-to-end manner from raw pixel images. CNN layers are firstly processed to generate middle level features. RNN layer is then learned to encode spatial dependencies. The C-RNN can learn better image representation, especially for images with obvious spatial contextual dependencies. Our method achieves competitive performance on ILSVRC 2012, SUN 397, and MIT indoor.

1. Introduction

In recent years, convolutional neural networks (CNNs) [23] have brought revolutionary changes to computer vision community. Upon the seminal paper [21], a number of recent works have been developed to further improve the representation power of CNNs in image classification [3,4,11,24,41,45], detection [10,14,34,40,52], face recognition [42, 46] etc. Most of these improvements focus on designing more sophisticated, deeper and wider networks, and aim to learn feature representations based on large and diverse datasets [5]. Besides CNNs, there are also some other deep neural nets, such as [29, 47, 54, 55], comparing with which CNNs are generally more efficient and effective.

The most important layers in CNNs are convolution layer and pooling layer. The convolutional layers convolve local image regions independently with multiple filters, and the responses are combined according to the coordinates of the image regions. The pooling layers summarize the feature responses, and pooling is processed with a fixed stride and a pooling kernel size. Both convolutional layers and pooling layers are performed without considering other regions. This setting has some obvious drawbacks. For example, when performing convolution/pooling for the top left image region, no matter what the appearance of the right bottom region is, the features of the top left region always remain the same. As a result, they fail to capture contextual dependencies for better representation.

In this paper, we propose to encode such correlations in image representation for better classification. But how to model the spatial dependencies? A straight-forward way is to learn all types of image region combinations. This will cost huge computation resources, and can hardly tolerate small part shifting or structure variance. The idealized network should have 'memory', and all the scanned area can be remembered and their spatial correlations can be analyzed. Recurrent Neural Networks (RNNs) [9, 17] are such neural networks developed for modeling the dependencies in time sequences. A RNN is a network which can have feedback connections among itself. Its hidden layer state is a function of the previous state, which can be further expanded as a function of all the previous states. Thus, RNN is inherently deep in time, and it is able to retain all the past inputs, based on which, the network can discover correlations between the input data at different states of the sequence.

However, in image domain, we cannot directly use the general RNNs, since 1) we do not have existing sequences, 2) the intermediate label of each state (image region) of the sequence is not given, only the label of the whole sequence (image) is available. For the first challenge, we pro-

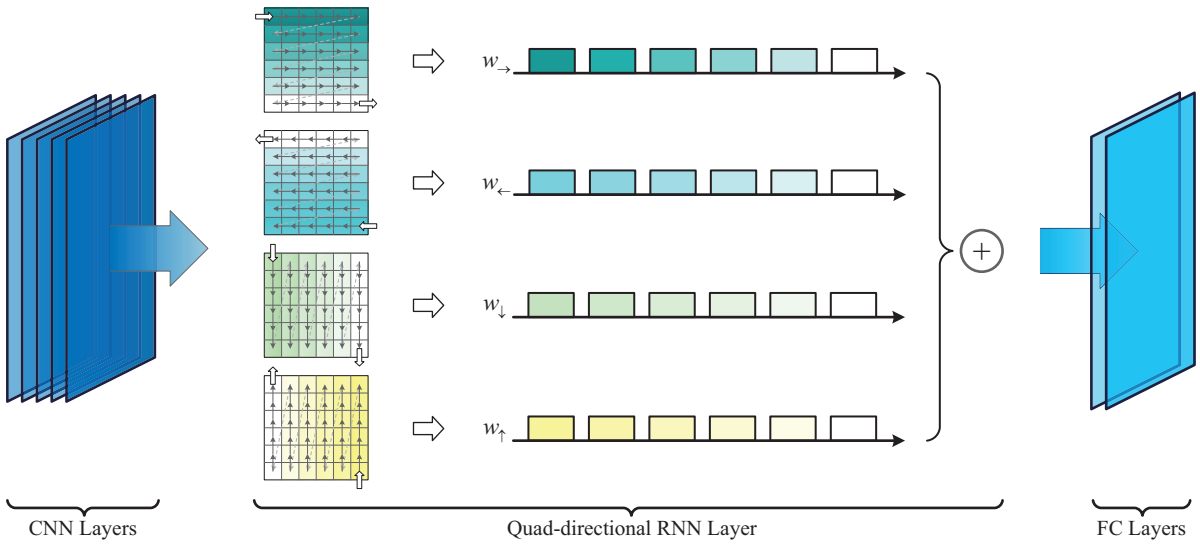

Figure 1: The overall C-RNN framework. **CNN layers:** The first five layers are convolutional and pooling layers, which aim to extract middle level image region representations. **RNN layers: 1)** The outputs of CNNs are parallelly converted into four spatial sequence by using four different directional scanning (left-to-right, right-to-left, top-to-bottom, and bottom-to-top). For each sequence, there is a scanning window with size of one row (or one column) of image regions, and the non-overlapping window keep scanning until the whole image sequence has been covered. **2)** For each scanning window, we focus on learning the local spatial dependencies of the segment of the sequential image regions (each region is represented as a feature vector), and update the shared unified RNN weights $W_{\rightarrow}$ (left-to-right), $W_{\leftarrow}$ (right-to-left), $W_{\downarrow}$ (top-to-bottom), and $W_{\uparrow}$ (bottom-to-top). The weights updating are sequentially processed window by window. **FC layers:** Gather the four directional RNN feature outputs and connect to two fully connected layers. (Best viewed in color)

pose to scan the image region by region from four directions (left-to-right, right-to-left, top-to-bottom, bottom-to-top), and convert each image into four independent image region sequences. For the second problem, we propose to utilize a global hidden layer to collect all the image region representations, and further connect to image-level labels. Afterwards, for each image region sequence, we are able to utilize RNN to learn their spatial dependencies, and consequently improve the classification accuracy.

By combining the CNN and RNN, we achieve our C-RNN framework as shown in Figure 1. C-RNN not only utilize the representation power of CNNs, but also employ the context modeling ability of RNNs. CNN layers can learn good middle-level features, and help our RNN layer to learn effective spatial dependencies between image region features. Meanwhile, the context information encoded by RNN can lead to better image representation, and transmit more accurate supervisions to CNN layers during back-propagation. Our C-RNN model can achieve competitive results on ILSVRC 2012, SUN 397, and MIT indoor.

2. Related Works

Recently, deep Neural Networks have made great break through in many computer vision areas in the last few years. There are a lot of successful deep frameworks, such as convolutional neural networks [3, 4, 7, 10, 11, 14, 21, 23, 24, 33, 40, 41, 45, 46], deep belief nets [15, 25, 32], and auto-encoder [1, 16, 50, 53], etc. Among all of them, CNNs are the most developed networks for image classification tasks. The key idea of CNNs is progressively learning more abstract (higher visual level) patterns: the first few layers learn 'Gabor like' low level features (e.g. edges and lines); the middle layers learn representation of parts of the objects (e.g. 'tire' and 'window' in the images with label 'car'); the higher layers learn the the feature of the whole image (e.g. image with label 'car').

In recent years, RNNs have achieved great success in natural language processing [2, 12, 20, 30, 31, 43, 44]. RNNs [8, 9, 17] are neural networks for modeling sequential dependencies. RNNs are networks with 'memory', they allow connections from the previous states to the current ones. Through these connections, the model can retain information about the past inputs, and it is able to discover correlations among the input data that might be far away from each other in the sequence. Because of the reuse of hidden layers, only a limited number of neurons need to be kept in the model, and testing is fast. Another advantage of RNN is the possibility to represent more advanced patterns of de-

pendencies in the sequential data.

However, RNNs are naturally suitable for dealing with 1D time sequences (e.g. text, speech), while images are 2D data, and the spatial context is undirected. In this paper, we convert the 2D images into 1D quad-directional spatial sequences by parallelly scanning the images from four different directions. Consequently, for each image region, the contextual information from all directions can be captured. RNNs have rarely been applied in computer vision. There are only few related works that partially utilize the recurrent idea. In [35], shared CNNs were used to model the long range pixel label dependencies by learning the connections between different scales of image patches. Different from this work, our RNN models the correlations between different image regions. Our tasks are also different from scene labeling, where all the pixels have labels, while in image classification, only the image level labels are given, thus our supervision is much weaker. Another work is DrSAE [37] which builds an auto-encoder with rectified linear units for digit number recognition. In DrSAE, iterative auto-encoders are used to encode the whole image. Different from them, we aim to model the spatial contextual dependencies of local image region features. In [13], MDLSTM was proposed to solve the handwriting recognition problem by using RNN. Different from this work, we utilize quad-directional 1D RNN instead of their 2D RNN, our RNN is simpler and it has fewer parameters, but it can already cover the context from all directions. Moreover, our C-RNN make both use of the discriminative representation power of CNN and contextual information modeling capability of RNN, which is more powerful for solving large-scale image classification problem.

3. Convolutional Recurrent Neural Network

Convolutional recurrent Neural Network (C-RNN) consists of five convolution layers, one recurrent layer, and two fully connected layers. Similar to the first five layers of the popular seven layer Alex-Net [21], the CNN layers are used to learn middle-level visual patterns. The RNN layer is employed to learn spatial dependencies between the middle level visual patterns. The final two fully connected layers are used to gather the RNN outputs and learn a global image representation. Afterwards, an N-way (N denotes the class number) softmax layer will be applied for classification. In the rest of this section, we will introduce our C-RNN in detail.

3.1. Convolutional Neural Network

As shown on the left part of Figure 1, we firstly utilize five convolutional layers to learn middle-level feature representations from raw pixel images.

Based on the visualization result in [51], when the number of convolutional layers increases, more abstract and robust patterns can be extracted. When being trained on ImageNet, the fifth convolutional layer can possibly localize parts and objects in the images. Thus, such CNN features are very suitable for representing the mid-level elements, based on which, we are more likely to learn appropriate spatial dependencies by using RNN, and further achieve better global image representations by connecting to two fully connected layers. With back propagation, the global representations can also transmit back supervisions for RNN to improve the spatial dependency encoding, and RNN can further help the CNNs to learn better middle-level and low-level features.

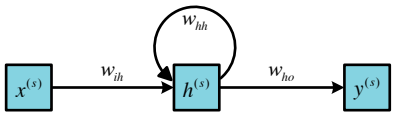

Figure 2: General RNN structure. In each state s of the sequence, there are two inputs for the hidden layer $h^{(s)}$: the current state input $x^{(s)}$, and the previous state hidden layer output $h^{(s-1)}$. The predicted output label $y^{(s)}$ dependents on $h^{(s)}$.

3.2. Recurrent Neural Network

Based on the fifth CNN layer, as shown in the central part of Figure 1, we build our RNN layer to model the spatial dependencies in the images.

Since RNN is originally designed for time sequences, thus it cannot directly be applied on images. We firstly convert the 2D images into four 1D image region sequences through four different scanning directions. Afterwards, we learn spatial context dependencies base on the sequences of image regions.

3.2.1 General RNN

RNNs are neural networks with feedback loops that connect the output of the previous hidden states to the current ones. They are designed to encode contextual information for sequences. In recent years, RNNs have shown good performances on natural language processing [12, 20, 31, 43]. In a typical recurrent network as shown in Figure 2, when processing a sequence with length S, its hidden layer feature $h^{(s)}$, and the predicted output label $y^{(s)}$ at the state $s \in [1, \cdots, S]$ can be represented as:

$$h^{(s)} = f_h\left(W_{hh}h^{(s-1)} + W_{ih}x^{(s)} + b_h\right) \quad (1)$$

$$y^{(s)} = f_o\left(W_{ho}h^{(s)} + b_o\right) \quad (2)$$

where $x^{(s)}$ is the s-th input data, $h^{(s)}$ denotes the hidden layer units, $y^{(s)}$ represents the output, W_{ih}, W_{hh}, W_{ho} are

the transformation matrices between $x^{(s)}$ and $h^{(s)}$, $h^{(s-1)}$ and $h^{(s)}$, $h^{(s)}$ and $y^{(s)}$. b_h and b_o are the constant bias terms, while f_h and f_o are the non-linear activation functions.

Because of the feedback loops, RNNs are able to remember all the previously processed data. The 'memory' is progressively increased by updating W_{ih}, W_{hh}. W_{ho}, and each state can be unfolded to a function of all the previous states. Ideally, the RNNs can remember whatever they have 'seen'. Such historical information can be used to find meaningful connections between the current data and its context. Here we utilize RNN to learn the connections between image regions at different spatial positions, which we called spatial dependencies.

3.2.2 RNN for Images

RNNs are naturally suitable for learning contextual dependencies. Similar to text and speech, image data also carries such structured dependencies.

However, in images, there are no existing sequences. It is difficult to build undirected graphs for image regions based on the existing CNN framework. Thus, we propose to convert each image into 1D spatial sequences, and use RNN to learn the spatial dependencies of image regions. Furthermore, the conventional single directional RNNs can only make use of the previous context, however, future context should also be helpful [39], especially in image data, where no fixed dependency direction exists. Therefore, we generate quad-directional spatial sequences to cover the context from all the image regions.

We use four different sequence scanner to convert each image into four spatial sequences: left-to-right sequence, right-to-left sequence, top-to-bottom sequence, and bottom-to-top sequence. As shown in the middle of Figure 1, taking the left-to-right sequence as an example, the sequence starts from the top left corner, and ends at the right bottom corner, the sequence is scanned from left to right, top to bottom, until the whole image has been covered. Similarly, we generate the other three sequences. In our C-RNN framework, each input data point corresponds to a 256-dimensional feature vector in the CNN response maps (one corresponds to an image patch in the original image, the data points correspond to overlapped neighborhood image patches [51]).

For each sequence, unified weights W_{hh} and W_{ih} ($W_{hh_\rightarrow}$ and $W_{ih_\rightarrow}$ for the left-to-right, $W_{hh_\leftarrow}$ and $W_{ih_\leftarrow}$ for the right-to-left, $W_{hh_\downarrow}$ and $W_{ih_\downarrow}$ for the top-to-bottom, and $W_{hh_\uparrow}$ and $W_{ih_\uparrow}$ for the bottom-to-top direction) are shared for all the image region features in each sequence.

Referring to Equation 1, for the spatial sequences, we can formulate our quad-directional RNN as:

$$h_{\rightarrow}^{(s)} = f_h\left(W_{hh_\rightarrow} h_{\rightarrow}^{(s-1)} + W_{ih_\rightarrow} x^{(s)} + b_{h_\rightarrow}\right) \quad (3)$$

$$h_{\leftarrow}^{(s)} = f_h\left(W_{hh_\leftarrow} h_{\leftarrow}^{(s-1)} + W_{ih_\leftarrow} x^{(s)} + b_{h_\leftarrow}\right) \quad (4)$$

$$h_{\downarrow}^{(s)} = f_h\left(W_{hh_\downarrow} h_{\downarrow}^{(s-1)} + W_{ih_\downarrow} x^{(s)} + b_{h_\downarrow}\right) \quad (5)$$

$$h_{\uparrow}^{(s)} = f_h\left(W_{hh_\uparrow} h_{\uparrow}^{(s-1)} + W_{ih_\uparrow} x^{(s)} + b_{h_\uparrow}\right) \quad (6)$$

$$h^{(s)} = h_{\rightarrow}^{(s)} + h_{\leftarrow}^{(s)} + h_{\downarrow}^{(s)} + h_{\uparrow}^{(s)} \quad (7)$$

where s is the index of the data point in the sequence, $x^{(s)}$ is the input data. $h_{\rightarrow}^{(s)}$ denotes the left-to-right hidden layer units, $h_{\leftarrow}^{(s)}$ denotes the right-to-left ones, $h_{\downarrow}^{(s)}$ denotes the top-to-bottom ones, and $h_{\uparrow}^{(s)}$ denotes the bottom-to-top units. $h^{(s)}$ is the combination of these four, which are the final feature outputs of the RNN layers.

3.2.3 RNN Optimization

Ideally, all of the regions in each sequence should be processed one by one, and the weights of RNN layer will be updated after the whole sequence has been processed. However, the gradients will vanish very fast in this way. According to the RNNs optimization notes in [30], RNNs can be simply and effectively optimized by back propagation through time (BPTT), which is a variant of back propagation for time sequences. In BPTT, the recurrent nets are unfolded to a limited size of the sequence, and converted into feed-forward networks with multiple deep layers. Consequently, traditional back-propagation for deep networks can be directly applied.

To make the RNN training effective and efficient, we unfold the feedback networks to feed-forward ones with limited length of sequence window. As show in the middle of Figure 1, for each sequence, there is a scanning window which covers n (n equals to the number of states in one row or one column) image regions. The window slides from the start to the end of the sequence in an non-overlapping way, and there are S/n windows (S is the length of the sequence) in each sequence. Thus, for each sequence, its shared weights (W_{ih} & W_{hh}) will be updated window by window for S/n times. In detail, for each window, only the image regions within the current window will be activated with the recurrent transformation, while for the other regions in the sequence, their previous states will be directly passed forward to help calculating the overall loss.

The RNN sliding window length is n, which means the spatial sequences should be unfolded for every n states. After the unfolding operation, the RNN equals to a feed-forward deep network with the depth of n layers (all the layers share the same weights). By utilizing the 'weight sharing' setting in Caffe [18], we can perform the BPTT

optimization for the shared RNN weights. In the rest of this sub-section, we will take the left-to-right sequence as an example, and describe the forward and backward procedures in the RNN optimization.

In the forward procedure, The Equation 3 can be unfolded as:

$$\begin{aligned} h_{\rightarrow}^{(1)} &= f_h\left(W_{hh_{\rightarrow}} h_{\rightarrow}^{(0)} + W_{ih_{\rightarrow}} x^{(1)} + b_{h_{\rightarrow}}\right) \\ h_{\rightarrow}^{(2)} &= f_h\left(W_{hh_{\rightarrow}} h_{\rightarrow}^{(1)} + W_{ih_{\rightarrow}} x^{(2)} + b_{h_{\rightarrow}}\right) \\ &\cdots \\ h_{\rightarrow}^{(n)} &= f_h\left(W_{hh_{\rightarrow}} h_{\rightarrow}^{(n-1)} + W_{ih_{\rightarrow}} x^{(n)} + b_{h_{\rightarrow}}\right) \end{aligned} \tag{8}$$

where $W_{ih_{\rightarrow}}, W_{hh_{\rightarrow}}, b_{h_{\rightarrow}}$ are the shared RNN weights and bias term, n is the length of the scanning window. If the scanning window is the first window of the sequence, $h_{\rightarrow}^{(0)}$ equals to zero, otherwise, it represents the last state of the previous window of the same sequence.

In the back-propagation procedure, the weights of each unfolded step can be updated step by step as follows:

$$\begin{aligned} W_{ih_{\rightarrow}}^{(s+1)} &= W_{ih_{\rightarrow}}^{(s)} + x^{(s)} e_{h_{\rightarrow}}^{(s)} \alpha \\ W_{hh_{\rightarrow}}^{(s+1)} &= W_{hh_{\rightarrow}}^{(s)} + h_{\rightarrow}^{(s-1)} e_{h_{\rightarrow}}^{(s)} \alpha \end{aligned} \tag{9}$$

where $e_{h_{\rightarrow}}^{(s)}$ is the gradient of error propagated from the output layer to the hidden layer at step s, α is the learning rate.

Similarly, forward and backward procedures of the other three directional RNN sequences (correspond to Equation 4, 5, and 6) can be achieved.

3.3. Fully Connected Layers

Different from general RNNs, which deals with the problems that have label $y^{(s)}$ for each $x^{(s)}$, we do not have any intermediate labels except the image label. Consequently, Equation 2 cannot be directly used in our C-RNN. Instead, we utilize the fully connected layers to collect all the hidden units in the RNN layers (Equation 7), and connect with the final image label. Similar to the fully connected layers in Alex-Net [21], we use two fully connected layers, and one softmax layer on the top:

$$\begin{aligned} g &= f_g(W_{hg} H + b_g) \\ y &= f_o(W_{go} g + b_o) \\ H &= [(h^{(1)})^T, \cdots, (h^{(s)})^T, \cdots, (h^{(S)})^T]^T \end{aligned} \tag{10}$$

in which, W_{hg} and W_{hg} are the fully connected transformation matrices. W_{hg} transfer the concatenated RNN outputs H to the global hidden layer g. While W_{go} transfer g to the predicted class label y. Particularly, H is the concatenation of all its sequential states $h^{(s)}$ ($s = 1, \cdots, S$). b_g and b_o are the bias terms, f_g is an non-linear activation function, and f_o is softmax.

4. Experiments

In this section, we will firstly describe the C-RNN network setting details, and then we will validate our C-RNN on three image classification benchmarks: ILSVRC 2012 [5], SUN 397 [49], and MIT indoor [36].

4.1. C-RNN Settings

We adopted the data prepossessing settings in Caffe [18]. For a training image, it was firstly resized to 256×256 pixels and subtracted by the pixel mean, based on which, 10 sub-crops of size 227×227 (1 center, 4 corners, and their horizontal flips) were extracted as the training data.

For the CNN layers, we followed the same filter settings in Alex-net [21]. The filter numbers (sizes) were: $96(11 \times 11)$, $256(5 \times 5)$, $384(3 \times 3)$, $384(3 \times 3)$, and $256(3 \times 3)$. For the first layer, the stride was 4, and the rest were 1. There were also three pooling layers for the first, second and fifth layer respectively, all of them were max pooling with the kernel size of 3×3, and the stride of 2. Thus, the size of feature response maps of the fifth CNN layer was $256 \times 6 \times 6$ (channel number $\times$ width $\times$ hight), which was the input to our RNN layer.

Since there are 6×6 output regions in the fifth CNN layer, thus, the length of our RNN sequence should be 36. For the RNN layer, we set the scanning window size as 6 (1×6 for one row or 6×1 for one column of regions), and there were 6 RNN windows for each of the four sequences. Where each region was represented as a 256-dimensional feature vector (channel number of the fifth layer CNN). Thus, the size of the RNN weight matrices $W_{hh_{\rightarrow}}$, $W_{ih_{\rightarrow}}$ (Equation 3); $W_{hh_{\leftarrow}}$, $W_{ih_{\leftarrow}}$ (Equation 4); $W_{hh_{\downarrow}}$, $W_{ih_{\downarrow}}$ (Equation 5); $W_{hh_{\uparrow}}$, $W_{ih_{\uparrow}}$ (Equation 6) were all set to 256×256. The non-linear transformation f_h, f_g, and f_o were all set to ReLU functions in our experiments.

The output unit number of the two fully connected layers were 4096.

Our basic C-RNN model was trained on the ILSVRC 2012 training set [5]. For the SUN 397 dataset, we fine-tuned the basic C-RNN model by replacing the 1000-way (ILSVRC 2012 contains 1000 classes) softmax layer with a new N-way (N is the number of classes in the dataset) softmax layer. The training batch size was 256, learning rate started from 0.01, momentum weight was 0.9, and both the RNN layer and the fully connected layers were applied with dropout rate of 0.5. All experiments were run with a single NVIDIA Tesla K40 GPU.

4.2. Experimental Results

4.2.1 Experimental Results on ILSVRC 2012

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) dataset [5] is the most prominent dataset for validating large-scale image classification algorithms. Most

Methods	test scales	test views	Top 1 val	Top 5 val
Alex-Net [21]	1	1	42.71%	20.01%
C-RNN	1	1	**41.85%**	**19.15%**
Alex-Net [21]	1	10	41.35%	18.74%
C-RNN	1	10	**40.16%**	**17.78%**

Table 1: Comparison of error rates on ILSVRC validation set.

of the existing deep neural networks depend on the large amount of training data provided by ImageNet. ILSVRC contains 1.2 million training images from 1000 different object centric classes. Here we evaluated our C-RNN and other CNN based networks on the validation dataset, which has 50,000 images in total.

In the upper part of Table 1, we compared our C-RNN with Alex-net [21] (we use the model released in [1]). Our C-RNN had the same settings with Alex-net, except that it directly connects the output of the fifth convolutional layer to the sixth fully connected layer, while our C-RNN uses the RNN to connect the fifth convolutional layer and the fully connected layers. Hence, the comparison with Alex-net can validate the effectiveness of our C-RNN. When we set all of the weight matrices W_{hh} to 0, and set W_{ih} to identity matrices, our C-RNN degenerates to the original Alex-net. In Table 1, we compared our C-RNN with Alex-net: testing on the eccentric view only or 10 test views, based on the Top 1 error rate and Top 5 error rate. The performance gain indicates that our C-RNN benefited from modeling the spatial dependencies by using RNN. Thus, besides going deeper and wider of the convolutional neural networks, RNN is another promising direction to build better neural networks for classification.

4.2.2 Experimental Results on SUN397

SUN 397 [49] is one of the largest datasets for scene image classification. Different from ILSVRC, it focuses on scene images rather than object images. It contains around 100,000 images from 397 different scene scenarios. We follow the general splittings in [49], which use 50 images per class for training, and 50 images per class for testing. Since there are a limited number of images in SUN 397, we pre-trained C-RNN on ILSVRC 2012, and then fine-tuned the network. For the Alex-net, we also applied fine-tuning for fair comparison, and we report the results before and after the fine-tuning.

As shown in the upper part of Table 2 (results were achieved with 1 test scale, 10 test views), our C-RNN performed much better than the Alex-net. Especially when the images have more obvious spatial layout, e.g. scene images, our C-RNN were more likely to capture such spatial correlations and learn better representations for scene images than the general CNNs. An interesting observation is that, when directly applying the model pre-trained on ILSVRC 2012, our C-RNN (43.38%) worked slightly worse than the Alex-net (43.74%), which was mainly caused by the image domain gap between SUN 397(scene dataset) and ILSVRC (object dataset). After fine-tuning on SUN 397, we were able to learn more data adaptive spatial dependencies and significantly outperform Alex-net with 4.02% in accuracy.

Methods	Accuracy
Alex-Net [21]	43.74%
Alex-Net [21] (finetune)	47.12%
C-RNN	43.38%
C-RNN (finetune)	**51.14%**
Xiao et al. [49]	38.00%
IFV [38]	47.20%
MTL-SDCA [22]	**49.50%**

Table 2: Comparison of accuracy on SUN 397 dataset.

In the lower part, we show the results of using shallow methods on SUN 397. Most of them heavily depend on combined hand-crafted features, and they usually have very high-dimensional features for the final image representation.

4.2.3 Experimental Results on MIT indoor

MIT indoor [36] is one of the most popular and challenging indoor scene image classification benchmarks. Different from the SUN 397 dataset, it focuses on indoor scenes, which tend to have more variations than outdoor scenes. MIT indoor has 67 classes. We followed the general splitting provided by [36], for each class, there are around 80 training images, and around 20 testing images. We also pre-train the C-RNN model on ILSVRC 2012, and fine-tuning the RNN layer and the 67-way softmax layer.

As shown in the upper part of Table 3 (results were achieved with 1 test scale, 10 test views), our C-RNN was able to achieve the accuracy of 65.07%. We could achieve

[1] https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

Methods	Accuracy
Alex-Net [21]	59.85%
Alex-Net [21] (finetune)	63.21%
C-RNN	58.96%
C-RNN (finetune)	**65.07%**
ROI + GIST [36]	26.50%
Object Bank [26]	37.60%
Visual Concepts [27]	46.40%
MMDL [48]	50.15%
IFV [19]	60.77%
MLrep + IFV [6]	66.87%
ISPR + IFV [28]	**68.50%**

Table 3: Comparison of accuracy on MIT indoor dataset.

the performance gain of 1.86% compared to Alex-net. The MIT indoor dataset only contains 5,360 training images, and the variance of image region dependencies are very large, thus, the representation power of C-RNN was not fully explored because of lacking of training data.

The bottom part of Table 3 shows the results of the state-of-art for non neural network methods. The most popular methods are combining middle-level feature (hand-crafted) with fisher vectors. Although very powerful on MIT Indoor, these methods cost much more computation power to perform middle-level patch searching and clustering, and the dimension of Fisher vectors is very high. In contrast, our C-RNN is an end-to-end feature learning method, the output feature is very compact (4096-dimensional), and the testing procedure is very efficient, which make it very appropriate for solving large-scale image classification problem.

4.2.4 Effect of Number of RNN directions

In this paper, quad-directional RNNs are built to encode the spatial contextual dependencies, but what are the benefits of doing so? In Table 4, we show the intermediate results of utilizing different numbers of context directions in the RNN layer of our C-RNN framework on ILSVRC 2012. On the left column of Table 4, the performance of using each single directional RNN is shown. On the right column of Table 4, the combination of two directional RNNs, and the complete quad-directional RNN results are given. We observe that utilizing one directional RNN only would already achieve satisfactory result, when activating multiple directions, especially the quad-directional RNNs, the performance gain was significant. Another observation is that the spatial correlation in the vertical direction (top-to-bottom & bottom-to-top) sequences are relatively more robust than the horizontal direction (left-to-right & right-to-left) sequences. The reason might be that the images are less sensitive to mirror flip (symmetrical about the vertical axis), while more sensitive to the flip about the horizontal axis.

Methods	Top 1 val	Methods	Top 1 val
C-RNN$_1(\rightarrow)$	40.89%	C-RNN$_2(\rightleftarrows)$	40.71%
C-RNN$_1(\leftarrow)$	40.86%	C-RNN$_2(\downarrow\uparrow)$	40.53%
C-RNN$_1(\downarrow)$	40.70%	C-RNN$_4$	40.16%
C-RNN$_1(\uparrow)$	40.74%		

Table 4: Error rates of applying RNN with different number of context directions on ILSVRC 2012. C-RNN$_1$ use one direction only; C-RNN$_2$ use the combination of two directions; and C-RNN$_4$ use the complete quad-directional RNN. (All results are achieved with 1 test scale, 10 test view)

4.2.5 RNN Complexity

In terms of the number of network parameters, each direction of the quad-directional RNN layer has two 256×256 weight matrices: W_{hh} and W_{ih}. Thus, there are 524,288 network parameters in the RNN layer in total. While in CNN, e.g. the second fully connected layer needs to learn a 4096×4096 weight matrix, which has 16,777,216 parameters. Thus, our RNN layer requires fewer parameters.

Furthermore, our RNN layer does not consume much memory. The only memory cost is from the intermediate hidden units $h^{(s)}$ (refer to Equation 7), which are 256-dimensional vectors. While in CNN, the most memory consuming layer is the first convolutional layer, which has 290,400 dimensional outputs. Thus, the RNN requires negligible extra memory.

5. Conclusions & Future Works

In this paper, we propose a new deep learning network C-RNN to encode spatial dependencies for image representation. CNN layers are firstly utilized to extract middle-level image region representation. Afterwards, our proposed quad-directional spatial RNN are employed to model the dependency correlations between different image regions. By combining the CNN and RNN, we are able to learn more powerful feature representations. Our C-RNN shows outstanding performance on all of the three image classification benchmarks.

In the future we would like to extend the RNN to deep RNN with recurrent hidden layers and enable the network to learn more complex spatial dependencies. Since the RNN is a general spatial dependency encoding layer, we would also try combinations of RNN with different types of deep neural networks other than CNN.

Acknowledgment

This research was carried out at the Rapid-Rich Object Search (ROSE) Lab at the Nanyang Technological University, Singapore. The ROSE Lab is supported by the National Research Foundation, Prime Ministers Office, Singapore, under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office.

This research is also supported by Singapore Ministry of Education (MOE) Tier 2 ARC28/14, and Singapore A*STAR Science and Engineering Research Council PSF1321202099.

References

[1] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. In *NIPS*, 2013. 2

[2] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012. 2

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014. 1, 2

[4] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012. 1, 2

[5] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 5

[6] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, 2013. 7

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014. 2

[8] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013. 2

[9] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. 1, 2

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[11] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*, 2014. 1, 2

[12] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014. 2, 3

[13] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 545–552, 2009. 3

[14] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 1, 2

[15] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 2006. 2

[16] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 2

[17] H. Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002. 1, 2

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 4, 5

[19] M. Juneja, A. Vedaldi, C. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *CVPR*, 2013. 7

[20] J. Koutník, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. In *ICML*, 2014. 2, 3

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2, 3, 5, 6, 7

[22] M. Lapin, B. Schiele, and M. Hein. Scalable multitask representation learning for scene classification. In *CVPR*, 2014. 6

[23] B. B. Le Cun, J. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990. 1, 2

[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014. 1, 2

[25] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, 2009. 2

[26] L.-J. Li, H. Su, L. Fei-Fei, and E. P. Xing. Object bank: A high-level image representation for scene classification & semantic feature sparsification. In *NIPS*, 2010. 7

[27] Q. Li, J. Wu, and Z. Tu. Harvesting mid-level visual concepts from large-scale internet images. In *CVPR*, 2013. 7

[28] D. Lin, C. Lu, R. Liao, and J. Jia. Learning important spatial pooling regions for scene classification. In *CVPR*, 2014. 7

[29] J. Lu, V. E. Liong, G. Wang, and P. Moulin. Joint feature learning for face recognition. *Information Forensics and Security, IEEE Transactions on*, 2015. 1

[30] T. Mikolov. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012. 2, 4

[31] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010. 2, 3

[32] V. Nair and G. E. Hinton. 3d object recognition with deep belief nets. In *NIPS*, 2009. 2

[33] M. Oquab, L. Bottou, I. Laptev, J. Sivic, et al. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014. 2

[34] W. Ouyang, P. Luo, X. Zeng, S. Qiu, Y. Tian, H. Li, S. Yang, Z. Wang, Y. Xiong, C. Qian, et al. Deepid-net: multi-stage and deformable deep convolutional neural networks for object detection. *arXiv preprint arXiv:1409.3505*, 2014. 1

[35] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014. 3

[36] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009. 5, 6, 7

[37] J. T. Rolfe and Y. LeCun. Discriminative recurrent sparse auto-encoders. *arXiv preprint arXiv:1301.3775*, 2013. 3

[38] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013. 6

[39] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997. 4

[40] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 1, 2

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2

[42] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. In *CVPR*, 2014. 1

[43] I. Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013. 2, 3

[44] I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted boltzmann machine. In *NIPS*, 2009. 2

[45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. 1, 2

[46] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014. 1, 2

[47] L. Wang, T. Liu, G. Wang, K. L. Chan, and Q. Yang. Video tracking using learned hierarchical features. *Image Processing, IEEE Transactions on*, 24(4):1424–1435, 2015. 1

[48] X. Wang, B. Wang, X. Bai, W. Liu, and Z. Tu. Max-margin multiple-instance dictionary learning. In *ICML*, 2013. 7

[49] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 5, 6

[50] X. Yan, H. Chang, S. Shan, and X. Chen. Modeling video dynamics with deep dynencoder. In *ECCV*, 2014. 2

[51] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013. 3, 4

[52] X. Zeng, W. Ouyang, M. Wang, and X. Wang. Deep learning of scene-specific classifier for pedestrian detection. In *ECCV*, 2014. 1

[53] J. Zhang, S. Shan, M. Kan, and X. Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *ECCV*, 2014. 2

[54] Z. Zuo and G. Wang. Learning discriminative hierarchical features for object recognition. *Signal Processing Letters*, 21(9):1159–1163, 2014. 1

[55] Z. Zuo, G. Wang, B. Shuai, L. Zhao, Q. Yang, and X. Jiang. Learning discriminative and shareable features for scene classification. In *ECCV*, 2014. 1