Deep Vision with Recurrent Neural Networks

Davit Buniatyan
BSc Computer Science, UCL
Submission Date: 26th April 2016

Supervisors

Lourdes Agapito
Adrian Penate-Sanchez

Abstract

In computer vision, pixel-depth estimation from camera is considered to be a basic problem. There has been vast amount of research and various sets of methods to tackle it from binocular or monocular, single images or image sequences desceribed in the following literature review. The limitations of methods include fixed or strictly moving camera, scene separation (outdoor or indoor), lack of dataset and more. Our research concentrates on depth reconstruction from monocular images using recurrent neural networks to capture the spatiotemporal context. In the beginning, we will concentrate on single image depth reconstruction. It is believed that the ability of human to reconstruct depth of a single image has statistical nature instead of geometrical interpretation.

Acknowledgement

I am thankful to my supervisor Lourdes Agapito and Adrian Penat-Sanchez, Graham Roberts (Tutor), Nicholas Leonard (RNN Torch package) and Corey Lynchs (2-LSTM), Nal and Alex Grave, Jan Medvesek and Emotech My family and my grandparents. , godfather girlfriend and Should be more thoroughly written. :)

Contents

1 Introduction

1.1 Motivation

Deep learning techniques such as Convolutional Neural Networks (CNN) achieved state-of-the-art results in Computer Vision and formulated branch in it Deep Vision in order to solve problems such as image classification, object detection and tracking, and semantic segmentation. However, these techniques often lack accuracy in low-level computer vision problems such as depth reconstruction that requires sequence-to-sequence pixel level labeling. RNNs are primarily used in modeling one-dimensional sequential inputs, however with the introduction of multi-dimensional RNNs and grid structure, which is a recent improvement, RNNs show state-of-the-art results in low-level Computer Vision tasks such as segmentation. The evaluation of recurrent architectures on spatiotemporal data is problematic because of limited training data.

The aim of this research is to evaluate the capabilities of Multidimensional Recurrent Neural Networks in depth reconstruction and propose modernized architecture that will take the advantage of both CNNs and RNNs and achieve real-time processing on spatiotemporal data, for instance, on image sequences (Video) or 3D environments (Virtual Reality). We will compare the model with widely used techniques, which have reached top-level results. Furthermore, in order to train and evaluate the algorithm due to the lack of efficient data we will generate synthetic realistic and non-realistic visual data using computer generated graphics and estimate the generalization feature of the model on real-life input. The goal of the research is to develop architecture that could be applied in robotics, for example, personal assistants or self-driving cars and enhance the performance of real-time computer vision by capturing spatiotemporal context.

1.2 Objectives

The aim of this research is to architect end-to-end novel Recurrent Neural Network for reconstructing the depth from sequence of images based on spatiotemporal context and achieve state-of-the-art results and real time computational complexity to provide industry ready solution. Furthermore, introduced architecture could be applied in other Computer Vision problems and introduce new approach in Deep Vision. This method is going to be unconstrained by the type of input.

The following objectives are going to be addressed.

1. How well the proposed architecture is able to extract long-term spatiotemporal context?

   We are going to estimate empirically the importance of spatiotemporal context in pixel level depth estimation and understand the significance of short (neighborhood pixels) and long (segment/object) term context.

2. To what extent do Multidimensional RNNs, trained on synthetic data, generalize on real-life input?

   As we are going to heavily base our training on synthesized data and validate the performance on real dataset, it is of vital importance to estimate the effectiveness of training recurrent neural networks on simulated environments and how well they generalize in real-life situations.

3. Complexity analyses of visual RNNs for parallel processing (Graphics Processing Unit, CUDA) on real-time applications.

   The significance of the research is to design an algorithm that could be applied in real life problems. Our objective is to keep the constraint on computational complexity as real-time as possible in order providing high applicability.

1.3 Applications

Primary applications of the solution are in the following fields,

- Robotics

  Even though RGBD cameras are expanding, they are still not cheap enough for

embedding them into robots. Depth reconstruction could be used as an alternative for environment reconstruction. They could be potentially used for enhancing gesture recognition and object detection in robots. Emotech, London-based startup, which develops personal assistant robot Olly, is interested in the proposed technology for enhancing computer vision abilities of their product [19].

- Visual FX

  In the movie production industry, software such as Boujou or Nuke have tools for cg artists to reconstruct scenes for post-processing, for instance, adding 3D characters into the scene or modifying objects. The limitation of the current software is that the primary usage is in outdoor static scenes. Real-time depth reconstruction will enhance artists' workflow not only in static but also in dynamic scenes.

- RGBD cameras

  Cameras such as Kinect lack accuracy in estimating the depth of far objects. The solution could be an additional online processing layer to improve the accuracy.

- Virtual Reality

  In order to enhance the virtual reality presence, one might propose 3D reconstruction of video content. The algorithm could be used to generate point cloud of movies.

1.4 Main Contribution

Primary applications of the solution are in the following fields,

- GridLSTM

  Even though RGBD cameras are expanding, they are still not cheap enough for embedding them into robots. Depth reconstruction could be used as an alternative for environment reconstruction. They could be potentially used for enhancing gesture recognition and object detection in robots. Emotech, London-based startup, which develops personal assistant robot Olly, is interested in the proposed technology for enhancing computer vision abilities of their product [19].

- Aplications

  In the movie production industry, software such as Boujou or Nuke have tools for cg artists to reconstruct scenes for post-processing, for instance, adding 3D characters into the scene or modifying objects. The limitation of the current software is that the primary usage is in outdoor static scenes. Real-time depth reconstruction will enhance artists' workflow not only in static but also in dynamic scenes.

2 Background

2.1 State-of-the-art

In the following review, we are going to discuss previous methods that tackled the problem from different perspectives under various contexts. We will discuss widely used benchmarks to compare results and introduce deep learning techniques that succeeded in similar problems such as estimation of optical flow or image segmentation. While there is a much prior work on estimating depth based on stereo images or motions there has been relatively little on estimating depth from a single image.

2.1.1 Single Images

One of the first methods to capture 3d structure of the scene was 'Tour into the picture' with spidery mesh interface that allowed doing perspective morphing from 2D images [1]. R. Szelisk and P. Torr introduced geometrically constrained methodology that has been foundation of approaches later defined in this review [2]. Another approach was extraction of the depth from Shading [3]. In this section, we will provide most descriptive methods that have been used to tackle the problem. It is generally considered to be ill posed problem, as there are infinitely many possible solutions.

Outdoor or Indoor [4, 5]

One of the first full automatic methods that creates directly 3D model from a single photograph is considered to be Automatic Photo Pop-up by D. Hoiem, A. Efros and M. Hebert at Carnegie Mellon University. The method is limited to dealing with only outdoor scenes such that it is possible to reconstruct a coarse, scaled 3D model from a single image by classifying each pixel as ground, vertical or sky and estimating horizon positions using features such as color, texture, image location and geometry. Later set of similar neighborhood of superpixels are considered to be constellations, such that one superpixel might belong to multiple constellations. Decision tree is trained for classifying each constellation ('ground', 'vertical' and 'sky') and thus each super pixel has a probability of being in each label. Then, the intersection of ground, vertical and sky lines are considered for defining the 3d orientation of planes in the image based on inferred camera properties. Then simple model is constructed using cutting and folding for generating those planes. The method achieves 30% accuracy for reconstructing accurate scenes.

Contrariwise, one of the first automatically 3d reconstructing algorithms from single indoor images is considered to be the method proposed by E. Delage, H. Lee and Andrew Y.Ng at Stanford that uses a dynamic Bayesian network model. Assuming 'floor-wall' geometry, the model takes as input: multi-scale intensity gradients in both the horizontal and vertical directions, as well as their absolute values and their squares, in addition to that, the neighborhood pixels, similarity to the floor chroma. Overall 50 features considered for estimating if the given pixel belongs to the floor or not. Using projective mapping, the 3d position of every pixel is located and accordingly walls reconstructing. The algorithm demonstrates the ability to detect robustly the floor boundary and generate accurate 3d reconstructions.

Learning Depth from Single Monocular Images [6]

Saxena introduced a method for predicting depth as a function of the image. The supervised learning method uses discriminatively trained Markov Random Field (MRF) that incorporates multiscale local and global features. The image is divided into small patches and single absolute and relative value is estimated for each patch. According to the paper monocular cues such as texture variations, texture gradients, occlusion, known objects sizes, haze, defocus are specific features that are good indicators of depth. Three local cues (texture variations, texture gradients and haze) are extracted as local representatives of patches (in total 17 filters). The sum absolute energy and sum squared energy is computed by multiplying the pixel intensity by the filter in the patch. To capture global information the same processing is done at different scales and neighborhood feature vectors are incorporated. For relative depth extraction, the feature vector is computed by histograms. As particular patch depends on the features of its neighbors and other parts, MRF is used to model the depth of the patch and depth of its neighboring patches. Gaussian and Laplacian MRFs are considered for maximizing log-likelihood. 425 image-depth map pairs are collected for training and testing the model by achieving average error 0.132.

Make3D [7]

Further on, Saxena extended the work to reconstruct 3D scenes from single still image using Markov Random Field to infer a set of plane parameters that capture both 3D location and 3D orientation of image depth cues as well as the relationship between different parts of image. The algorithm starts by segmenting the image into many small planar surfaces. Planar surface in their turn are segmented into superpixels with high probability of being on the same real plane. The features such as relation of the depth between superpixels, neighboring structures, Co-planar structures and co-linearity are captured in MRF model considering their 'confidence' level. For parameter learning

Multi-Conditional Learning was used where the graphical model is approximated by product of several marginal conditional likelihoods and plane parameters of 3d location and orientations are estimated. Totally 534 images-depthmaps have been collected and achieved 64.9% qualitatively correct 3D models by outperforming other methods and becoming state-of-the-art technique in 3D scene reconstruction. Furthermore, the model is extended to capture full photorealistic model based on multiple images.

From Semantic Labels [8]

B. Liu proposed a method for single image depth estimation from semantic labels. The work addresses the problem of depth perception from a single monocular image through the incorporation of semantic information. The algorithm has two phases. The first phase solves the problem of multi-class image labeling. Pixels are labeled with general categories (for example: sky, tree, road etc.) similar to outdoor/indoor scenes. The second phase is to learn the depth estimations for each semantic class based on each location. For example if the label is sky then the pixel is infinitely. Embedding several more precise assumptions (for example the class 'road' and 'ground' are usually on the horizontal plane or it more probable that 'road' and 'building' are near to each other) into Markov random field the depth map from single image is estimated. The model was trained on Saxena's outdoor dataset and achieved state-of-the-art results on $\log_{10}$ metric and comparable performance to state-of-the-art for the relative error metric.

Depth Fusion [9,10]

By the availability of cloud computing and 3D content, J. Konrad purposed automatic method for 2D-to-3D conversion based on the dataset of existing stereoscopic videos such as Youtube 3D. Taking an assumption that there exists similar stereo pair in the dataset given arbitrary image, the method extracts depth information of the pair and uses for estimation of the input image and generates stereoscopic view. In details, the algorithm does, k nearest-neighbours (kNN) search to find k most similar 2D left images in this case from Youtube 3D, estimates the parameters for warping the query image to the left image of each kNN stereopair, warps extracted disparity of video pairs to the query images, does median filtering of the k warped disparity field including cross-bilateral smoothing and generates the right image.

The approach later was reconsidered on extracting depth information from still images, such that given an image the kNN searches over either stereopairs or image+depth repository, fuses the depth maps (instead of disparity) and similarly filters. At the end, the algorithm renders the stereo image based on median field followed by suitable processing of occlusions and newly exposed areas. The following approach was

considered with Make3D and outperformed under indoor scenes, even though Make3D was originally trained on outdoor scenes. The training data included 1449 pairs of RGB and depth collected on Kinect camera.

Pulling things out of Perspective [11]

The paper mostly discusses about the significance using the properties of the perspective geometry that reduces the learning pixel-wise depth classifier to much simpler classifier predicting only the likelihood of a pixel being at an arbitrarily fixed canonical depth. It takes the advantage over other methods by constructing an image pyramid and in the result avoiding a pitfall, which requires the training set to have the same object at different depths in order to predict accurately. It states that that for stereo and multi-view images join semantic segmentation and 3D reconstruction leads to better results than performing each task separately.

2.1.2 Images Sequences

Recovery from video [12]

G. Zhang introduces a novel method for automatically constructing view-dependent depth map per frame by making depth values in multiple scenes consistent and assigning distinctive depth values for pixels. In order to meet objectives of consistency and distinction, the algorithm uses segmentation as the initial step and then iteratively refines disparities in a pixelwise manner. More precisely, the algorithm estimates depth maps for each pixel by belief propagation, incorporates segmentation, then initialize disparities with segmentation and plane fitting using a nonlinear continuous optimization, and refine the final output with bundle optimization. Even though lack of comparison with other algorithms, it was evaluated qualitatively and quantify and achieved visually very appealing results. However, if there is no sufficient camera motion, the recovered depths could be less accurate.

DepthTransfer [13]

Similar to Konrad's approach, depth extraction from video was proposed which is called DepthTransfer. The algorithm, given a database RGBD images, finds the similar images to the input image in RGB space. Then, found images and their depths are warped in order to align with the input image. Finally, an optimization procedure is used to interpolate and smooth the warped candidate depth values that result in inferred depth. In order to effectively find similar images their high-level image features are extracted and top 7 similar frames are selected for warping. The optical flow with motion

estimations is used for consecutive frames depth maps to be matched continuously. Separate dataset is collected for training the algorithm and results outperformed both DepthFusion and Make3D on single images on Make3D dataset and outperformed Depth Fusion train and tested on NYU Depth dataset even though the results are not that perceptual. According to the paper, generalization to interior scenes is much more difficult than outdoor.

Intrinsic Depth: Improving Depth Transfer with Intrinsic Images (will be added)

Photo Tourism: Exploring photo collections in 3D (will be added)

2.2 Deep Learning

As Convolutional Neural Networks (CNN) proved to be successful method for the image feature extraction and, hence, was widely used in current the state-of-the-art solution in various computer vision applications most notably object detection and scene classification. Here we present state-of-the-art models for depth estimation from single images and other applications that CNNs have been particularly useful.

Multi-Scale Deep Network [14]

A novel approach has been developed due to increase of Neural Networks activity in Computer Vision tasks that estimates the depth from single image directly regressing on the image. The method has two core components. The first component captures global structure scene and the second captures the local context. The goal of global structure is to predict overall depth map and embed into the local component. It takes down scaled image and passes through 4 convolutional and 2 fully connected layers. The local structure initially processes the image through a convolutional layer and concatenates with output from global component. Finally, after two convolutional layers the final depth image is computed. In addition to NYU and Make3D dataset the, at that time the method achieved state-of-the-art performance on KITTY dataset.

Convolutional Neural Fields [15]

Deep Convolutional Neural Field has been proposed for depth estimation using CNN as a feature descriptor based on Multi-Scale Deep Network and continuous Conditional Random Field (CRF) as loss layer, which is minimized with negative log-likelihood. The model requires segmented image of superpixels to feed to a unary layer and a pairwise layer. Both then outputs to CRF layer. Unary layer is a CNN with 5 convolutional and 4 fully connected layers that outputs n-dimensional vector contained regressed depth

values of the n superpixels. Simultaneously, pairwise layer takes similarity of all neighboring superpixel pairs as input, feed each of them to a fully-connected layer and outputs a vector containing all the 1-dimensional similarities. This is by far the most advanced method that uses deep learning and outperforms state-of-the-art methods in single image depth estimation trained and tested on NYU v2 and Make3D dataset.

FlowNet (will be added)

AutoEncoder

A neural network called autoencoder is used for dimensionality reduction of information trained on the input, itself. The hidden layer is contains less cells than the input/output layer. Hence, by minimizing the loss, useful features are preserved and encoded in hidden layer. K. Konda and R. Memisevic introduced unsupervised deep learning model using autoencoder that captures the depth and motion from sequence of stereo image pairs [16]. They extended autoencoder model by taking two separate inputs of the stereo image patches and concatenating into one hidden layer. As depth/motion recovery is usually a middle step for further problem solving, action recognition has been successfully attempted using extracted features from autoencoder, which resulted in state-of-the-art performance in 3D activity analysis.

2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) is a type of neural network that processes sequences one step at a time by passing the hidden state of the previous timestep to the next one and hence generates output taking into consideration context. RNNs are used primarily for many-to-one (when the sequence of information is parsed and final single output is made), one-to-one (at each timestep RNN produces one output) and many-to-many (all data is processed) labeling types [17]. Recurrent Neural Networks achieved successful results on language modeling, translation and character recognition. Simple RNN with unit activation cell suffers from vanishing-gradient problem. The hidden state information either diverges or converges to zero. Long-Short Term Memory cells has been proposed by S. Hochreiter and J. Schmidhuber [18] to capture long term dependencies by learning when to remember and forget information. LSTM is based on four cells called gates: input, memory, forget, output. Another limitation of RNN is that the model has only past context at each timestep. Bi-directional RNN has been proposed which consists of simultaneous layers which processing the input from first to last and from last to first correspondingly and then concatenates the output to perform prediction with past as well as future context [19]. RNNs have been particularly useful in image or video captioning [20, 21] for generating descriptions after preprocessing of

convolutional layers, however in order to be useful for more structured input in computer vision 1-dimensional limitation should be extended.

2.3.1 Multidimensional [22]

Alex Graves proposed multidimensional Recurrent Neural Network that extends temporal RNNs onto n-dimensional space. At each time-step the architecture takes an input of multi-dimensional data such as the pixel of image, video or MRI and the hidden state of the previous steps across all dimensions. The advantage of this model is that it scales linearly as the number of inputs grows and it is applicable to capture the long-term context of the data contrariwise to convolutional neural networks. As in bidirectional RNN model where the past and the future is encountered in the output by two simultaneous layers that parse the data from both sides at the same time, Graves introduced multi-directional model where the data is parsed from all possible dimensions. The model was tested on MNIST dataset. It showed equal results on the dataset, however outperformed state-of-the-art method CNN modeling in modeling image warping. Recent successful applications of multi-dimensional RNNs are presented below.

- **Scene Labeling [23]**

  Complete end-to-end learning based approach using 2D LSTM was proposed for scene labeling task. The network consists of 3 layers: input, hidden and output. The input layers takes RGB pixels (3xnxn) with sliding window, processes the information using 4 stacked multidirectional multidimensional layer to capture whole context and passes to the next two consecutive LSTM hidden layers at each timestep, which produces final result with softmax output layer. The model is the state-of-the-art technique for Stanford Background and Sift Flow dataset by outperforming other methods that embed CNNs. Even though the training takes 10 days on single CPU, the processing of each image is linear and takes 1.3s. Implemented on GPU. In conclusion, this paper supports the argument that RNNs are capable of handling pixel-level labeling tasks by capturing both local and global context.

- **Pixel Recurrent Neural Network [24]**

  Recent paper by Google DeepMind for modeling the distribution of natural images for predicting the image used enhanced version of 2D LSTM to recover unseen image parts without external information. It has been shown that PixelRNNs significantly improved results on the Binary MNIST and CIFAR-10 datasets. Furthermore, they

introduced ImageNet as a new dataset. And final conclusion has been reached that the PixelRNN is capable of modeling spatially local and long-range correlations and are able to produce images that are sharp and coherent

2.3.2 Grid-LSTM [25]

Furthermore, Nal Kalchbrenner, Ivo Daihelka and Alex Graves extend multidimensional capability of Recurrent Neural Networks by introducing Grid Long-Short Term Memory, a network of LSTM cells arranged in a multidimensional grid that can be applied to vectors, sequences or higher dimensional data such as images. Grid-LSTM outperforms Multidimensional by resolving instability of large grids by adding cells along the depth dimension. Additionally unit cells are simplified and unified such that from all dimensions the input and output consists of two gates: the memory and hidden gate. Each block is consisted from LSTM cells along each dimension. The network achieves state-of-the-art results on Wikipedia character prediction benchmark among neural nets and outperforms a phrase-based reference system on a Chinese-to-English translation task. Additionally achieves near state-of-the-art results on MNIST dataset.

3 Implementation

3.1 Theory

3.1.1 Long-Short Term Memory

Simple recurrent neural network computes output by taking previous output **h** and current input **x**. To solve vanishing gradient problem, usual LSTM cell along with input **x** and **h** previous hidden states, also takes **m** memory data from previous time step. Intuitively, LSTM learns when to remember and forget information.

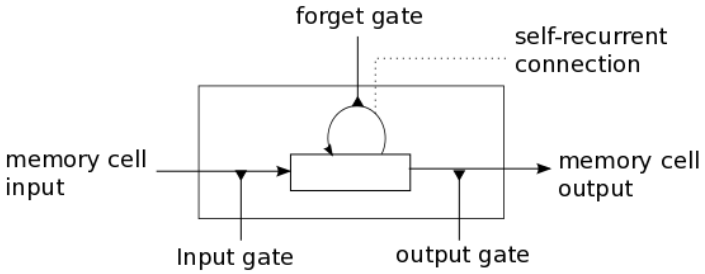

Figure 1: LSTM cell

LSTM model contains an input gate, a neuron with self-recurrent connection, a forget gate and an output gate.[1] Self-recurrent connection is responsible for storing the information of previous time step. Input Gate alters the memory state of the current cell and output gate affects future cells. Forget Gate decides when to remember or forget the previous information. The computation is done as follows.

$$\begin{aligned}
\mathbf{g}^{u} &= \sigma(\mathbf{W}^{u}\,\mathbf{H}) \\
\mathbf{g}^{f} &= \sigma(\mathbf{W}^{f}\,\mathbf{H}) \\
\mathbf{g}^{o} &= \sigma(\mathbf{W}^{o}\,\mathbf{H}) \\
\mathbf{g}^{c} &= \tanh(\mathbf{W}^{c}\,\mathbf{H}) \\
\mathbf{m}' &= \mathbf{g}^{f} \odot \mathbf{m} + \mathbf{g}^{u} \odot \mathbf{g}^{c} \\
\mathbf{h}' &= \tanh(\mathbf{g}^{o} \odot \mathbf{m}')
\end{aligned} \tag{1}$$

Where $\mathbf{H} = [\mathbf{I}_x, \mathbf{h}]$, **h** is the hidden output of previous cell, **m** is the memory output and $\mathbf{W} = [\mathbf{W}^u, \mathbf{W}^f, \mathbf{W}^o, \mathbf{W}^c]$ is the weight matrix to train. (σ - sigmoid function, tanh - tangent function and $\odot$ component wise multiplication)

Variation of LSTM called Gated Recurrent Unit (GRU) was proposed that achieves slightly better performance. Using same fashion of connecting gates or introducing new inner cells,

[1] http://deeplearning.net/tutorial/lstm.html
[2] https://moodle2.cs.huji.ac.il/nu15/pluginfile.php/316969/mod_resource/content/1/adam_pres.pdf

genetic algorithm generated three mutations (JZ1, JZ2 and JZ3), which outperform LSTM and GRU. For simplicity we will concentrate the scope of our project only on LSTM cells.

3.1.2 GridLSTM

Grid Long Short-Term Memory is a recurrent neural network of LSTM cells arranged in multidimensional grid that processes input such as vectors, images videos or MRI scans. Like multidimensional recurrent neural networks, model takes an input a data from the input dimension and hidden outputs of previous cells per each sequential dimension. At each step, each cell produces an output along each dimension for next steps including the final output.

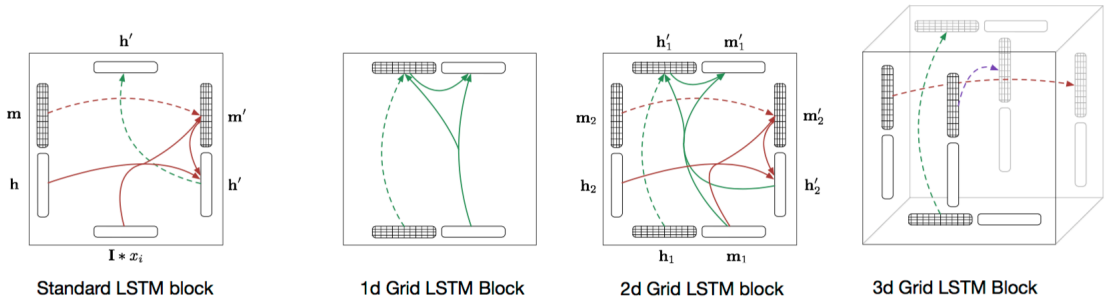

Figure 2: GridLSTM block

Unlike RNN or LSTM, GridLSTM cell called block takes from each dimension ($\mathbf{h}'_k$, $\mathbf{m}'_k$) (including input) and correspondingly processes outputs. For each k of n dimensions, block computes.

$$(\mathbf{h}'_k, \mathbf{m}'_k) = \text{LSTM}(\mathbf{H}, \mathbf{m}_k, \mathbf{W}_k) \qquad (2)$$

Where $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n]$, $\mathbf{h}$ is the hidden output, $\mathbf{m}$ is the memory output and $\mathbf{W} = [\mathbf{W}^u, \mathbf{W}^f, \mathbf{W}^o, \mathbf{W}^c]$ is the weight matrix to train. LSTM function is just the concatenation of formulae described in (1).

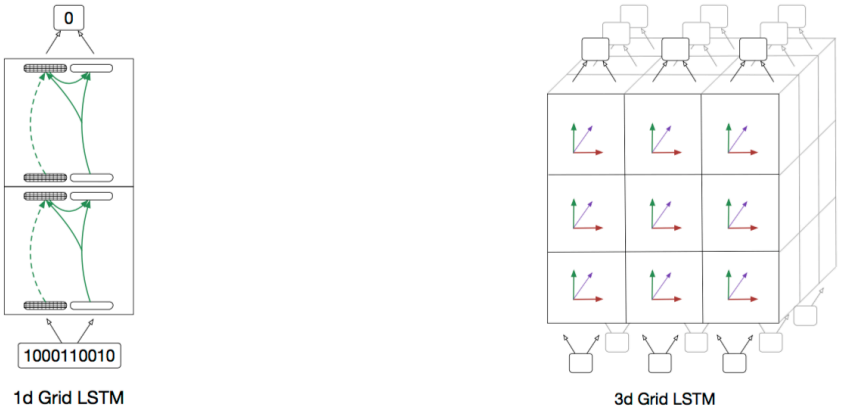

Figure 3: GridLSTM

Nal Kalchbrenner also introduced several additions to GridLSTM: priority dimensions and

weight sharing. In order to take processed information outputs from other dimensions to include into priority dimension, the cell computes initially outputs of secondary dimensions and only then concatenates with input of primary dimension and forwards the data. The weights of LSTM along one dimension could be shared across different layers.

3.2 Deep Learning with Torch

3.2.1 Torch Framework

Torch deep learning framework is used to build and train the model. Any Neural Network can be implemented using Torch modules with nn and nngraph packages by creating computational graph. Custom modules extend nn.module and can be easily plugged into any architecture. For developing your own module, a computational node should be created by defining forward and backward passes including calculation of gradient of the input.

Recurrent neural networks are unfolded at each time step of input sequence. Thus, a clone of RNN cell is created in order to suit into nn.graph abstraction. In other words, RNN is abstracted into usual Neural Network that takes a sequence at once and outputs another sequence. Furthermore, there exists RNN package that takes the responsibility of cloning by introducing Sequencer decorator. Any RNN cell (including simple sigmoid cell, LSTM or GRU) extends AbstractRecurrent class and is decorated by Sequencer to fit into any NN architecture.

To extend AbstractRecurrent class in order to define custom module mainly three functions should defined

buildModel() - builds the computational graph/model

updateOutput(input) - forwards the input and returns the output

updateGradOutput(input, gradOutput) - proceeds the backward pass and returns the input gradient

In the following part our modular implementation of 3D Grid LSTM is discussed. It is based on Corey Lynch's 2D GridLSTM code of character language model that is in turn based on Andrej Karpathy's Char-RNN.

3.2.2 Build

As RNN model should be unfolded in Torch abstraction, 3D GridLSTM takes 6 inputs ($\mathbf{m}_\mathrm{I}$, $\mathbf{h}_\mathrm{I}$, $\mathbf{m}_\mathrm{x}$, $\mathbf{h}_\mathrm{x}$, $\mathbf{m}_\mathrm{y}$, $\mathbf{h}_\mathrm{y}$) where first 2 vectors represent the input and the last 4 vectors represent hidden previous timesteps along x and y dimensions. Input sums of **h**s are correspondingly calculated and passed through three LSTM function to process outputs.

$$(\mathbf{h}_{\mathrm{output}}, \mathbf{m}_{\mathrm{output}}) = \mathrm{LSTM}([\mathbf{h}_\mathrm{I}, \mathbf{h}_\mathrm{x}, \mathbf{h}_\mathrm{y}], \mathbf{m}_\mathrm{I}, \mathbf{W}_\mathrm{I})$$

$$(\mathbf{h}'_x, \mathbf{m}'_x) = \text{LSTM}([\mathbf{h}_I, \mathbf{h}_x, \mathbf{h}_y],, \mathbf{m}_x, \mathbf{W}_x)$$
$$(\mathbf{h}'_y, \mathbf{m}'_y) = \text{LSTM}([\mathbf{h}_I, \mathbf{h}_x, \mathbf{h}_y],, \mathbf{m}_y, \mathbf{W}_y)$$

And in case of enabling priority dimensions the final output of the block is

$$(\mathbf{h}_{\text{output}}, \mathbf{m}_{\text{output}}) = \text{LSTM}([\mathbf{h}_I, \mathbf{h}'_x, \mathbf{h}'_y], \mathbf{m}_I, \mathbf{W}_I)$$

Tying the weights includes creating shared global variable and reusing it.

3.2.3 Forward

In order to comply with RNN Torch abstraction, any input is treated as one-dimensional sequence with N dimensions reshaped to one. So, at any time step only one input is given with an order based on reshaping policy (For example for 2D case: row-wise or column-wise). Hidden previous states are zeroed at the boundaries of the image (or any other input) when there is no prior information available. During the forward pass, along with the input, the algorithm concatenates previous outputs of block along each dimension and feeds into the network. Using computational graph defined in Build section, Torch computes the output of the given input at the time step and stores all outputs while passing only main dimension to the next layer as the output of neural network. Stored outputs are retrieved for next time steps. Hence network remembers the previous input.

3.2.4 Backward

To train the network, Backpropogation algorithm is used. The *updateGradInput*() function is defined in order to estimate the gradient of the module with the respect to its input. We parse the sequence in reverse order, being output aware. In other words, the layer knows all its outputs for the given input as it assumes that forward pass has been already processed. The gradient of the output is already estimated by loss criterion, however gradients along inner dimensions are retrieved from previous gradient calculations (which are stored in *drnn-states*). So, backpropogating the computational graph by providing the input and gradients with the respect to its output, we recieve the gradient with the respect to its input. The gradient of the module with the respect to its own parameters is also processed and finally the weights of GridLSTM are updated. And thus the network 'learns'.

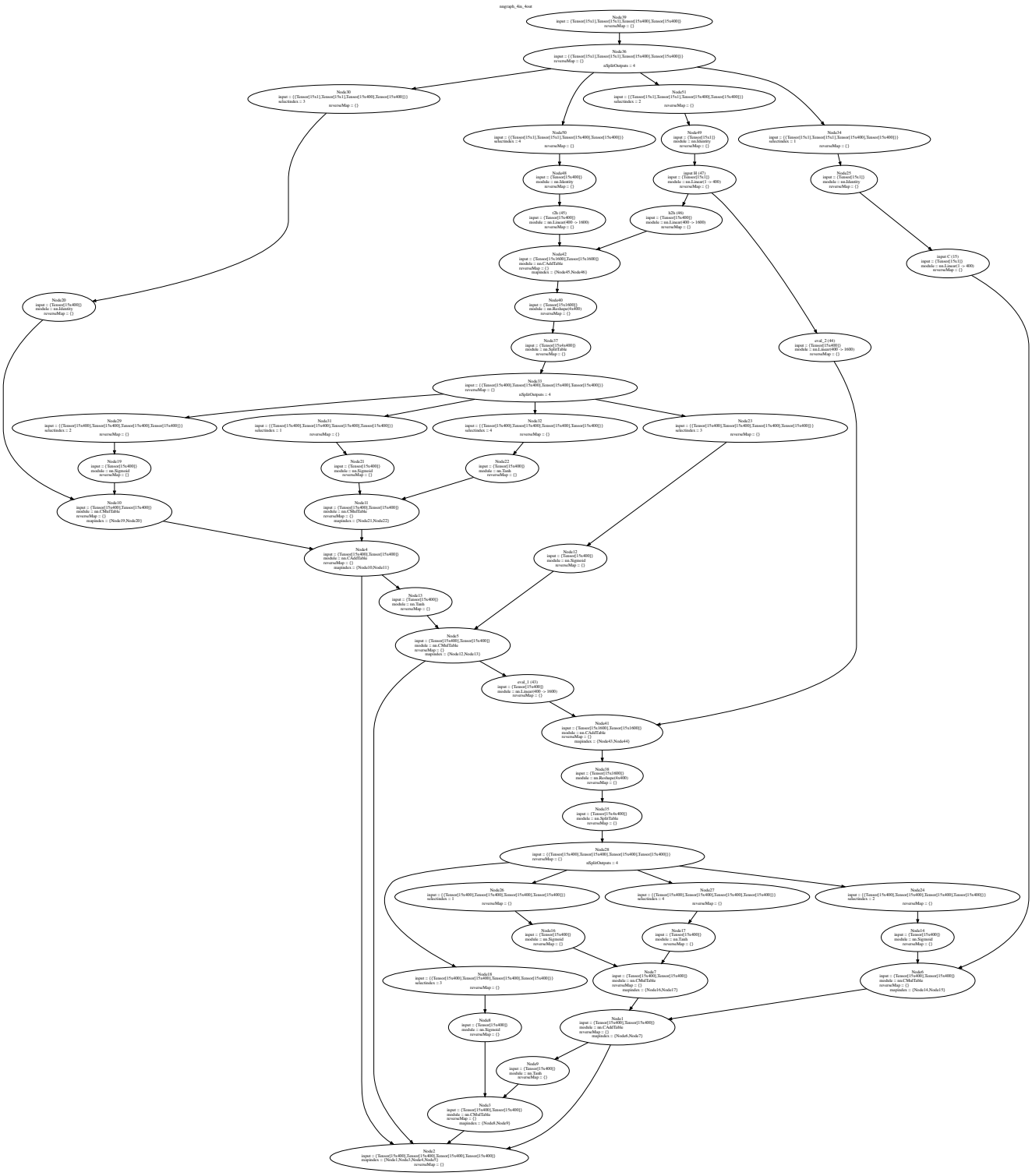

Figure 4: Implementation architecture of GridLSTM build. where described the computational graph of nn package in torch.

3.2.5 Training

To train the network, each node of the network is back propagated and correspondingly the weights are updated. User defines the criterion of loss, which estimates how good is the prediction of the network based on the ground truth. There are built-in criterions such as bi as binary cross-entropy, negative log-likelihood, absolute criterion, however one can define their own criterion as a module under Torch NN abstraction. For GridLSTM, as described in the paper, instead of using Stochastic Gradient Descent (SGD) learning algorithm, the module is usually trained using a stochastic optimization called Adam by taking advantage[2] of effectiveness on sparse gradients and non-stationary objectives based on adaptive estimates of low-order moments[3]. The method of training and criterion varies for each type of problem.

3.2.6 Evaluation

In order to provide initial evidence that the model is actually working, initially prototype of 2D GridLSTM has been built for parsing 1D dimension data. We replicated an example in the paper that sums two numbers. The input of the network is a sequence of two 15-digits separated by GridLSTM. It outputs another sequence of 15 or 16 digits. For testing dataset random 100 numbers are generated and digit error is calculated. The mini-batch size 15 and the algorithm uses Adam optimization method with 0.001 learning rate. In the figure 5 and 6 below, a comparison of digit-wise accuracies is provided, where one can see that GridLSTM (green) slightly outperforms usual LSTM in stacked in 2 layers.

Figure 5,6: Comparison of 2D GridLSTM and LSTM on Sums digits.

Due to our computational limitations it is not sufficient to validate the model, as in order to reach 100% accuracy it is required to train the model for up to 5 million

[2] https://moodle2.cs.huji.ac.il/nu15/pluginfile.php/316969/mod_resource/content/1/adam_pres.pdf
[3] http://arxiv.org/pdf/1412.6980v8.pdf

iterations.

Also we tried to compare the Lynch's available implementation on character level language modeling, however have been unable to set up the environment because of architectural differences. Their model is solely based on nngraph and it will be very time consuming for reintegration and synchronization with our model.

Another 'unit-test' for neural networks is considered to be over fitting the model as soon as possible and feed the same training set back to reach as good as possible. The given technique was used during the development of the model.

In conclusion, even though the lack of significant results that would describe evidence that the model is correct, we have been able to use validation techniques to provide sufficient evidence that the given network is actually learning and is better than simple LSTM cell in the given example. In order to validate the model, we will fully replicate digit classification for images.

3.3 Directional Layers

As Recurrent Neural Networks parse a sequence iteratively, the hidden input and memory input from previous time steps contain only past information. In other words the RNN remembers only what it has seen. However, it is often the case when presented information has no actual beginning. It either was captured at once or contains information that is based on not only past context but also future context. We will describe directional layers, which overcome this limitation of RNN.

3.3.1 Bi-Directional

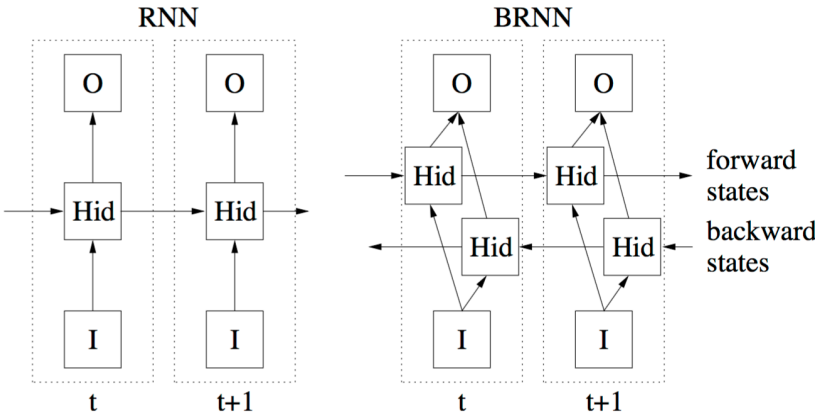

Figure 7: Comparison of simple Recurrent Neural Network and Bidirectional Recurrent Neural Network

The main motivation behind Bi-Directional RNN is that while making prediction at each time step, the output should be based not only on past, but also future context. Extending simple one directional recurrent neural network by introducing parallel layer that parses the input from negative direction solves the issue. As depicted in Figure 7, forward layer processes the information from left to right and backward layer processes the original sequence from right to left. Final composition happens on output layer by concatenating forward and backward outputs at each state.

```
Backward:add(nn.ReverseTable())
Backward:add(nn.RNN(...)))
Backward :add(nn.ReverseTable())
```

On implementation side, reversing the input sequence, applying usual RNN layer and then reversing back the output to the original order implements backward layer, which is processed simultaneously with original RNN layer. The final layer usually uses either sum or multiplication concatenation function to process the output.

3.3.2 Multi-Directional

When the input is not one dimensional, a question arise how to treat the past/future context at any coordinate in space. Having the same intuition behind, Alex Grave at al introduced multidirectional RNN where multidimensional input is parsed from each corner (vertex) of the input.

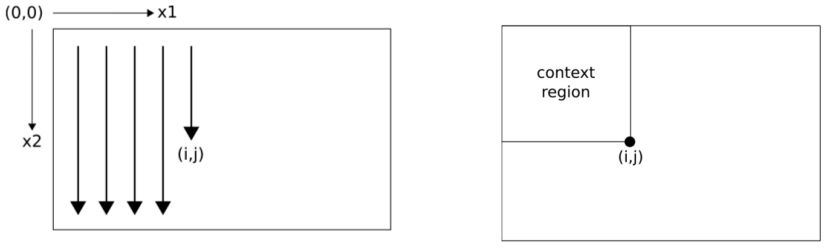

Figure 8: One directional parsing of an image (left) and available context at each timestep (right)

For N dimensional input, parallel 2^N layers are added to the network each parsing the input sequence from each vertex. In Figure 9, a 2d image is shown where all contexts at each time step is available for producing the output.

As we simplify N dimensional input in our implementation of GridLSTM by reshaping into 1D sequence under AbstractRecurrent class, more complicated reversing algorithm should be proposed. For 2d images, simple layer is developed that outputs the

symmetric image represented in one dimensions. This simulates top-right corner

```
Layer_3:add(nn.SymmetricTable(width, height))
Layer_3:add(nn.rnn(...))
Layer_3:add(nn. SymmetricTable (width, height))
```

Using the same fashion, we add ReverseTable layer, which will simulate the opposite left-bottom corner and in the result we will have all four layers for providing context-wise predication at each pixel. Later, it will be seen that actually in digit classification problem described in GridLSTM paper, instead of simultaneously using the layers, they are used in sequential order, which achieves better results in the given problem.

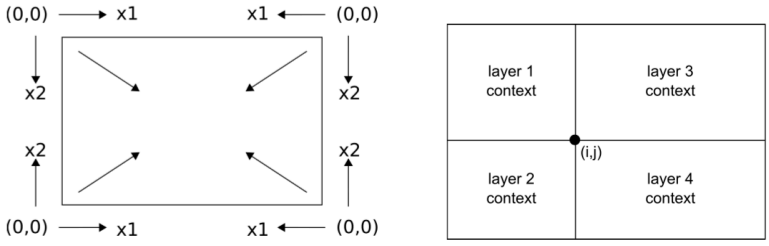

Figure 9: Multi directional parsing from each corner (left) and the available context at each timestep

3.4 Digit Classification

Digit Classification on MNIST dataset is considered to be 'hello world' for machine learning models. It has been benchmarking environment for testing any new deep learning algorithm or architecture. Multi layer Convolutional Neural Network (CNN) achieved near-human performance on digit classification with error of 0.23 percent[4], which is the tate-of-the-art at the current moment. GridLSTM achieved near state-of-the-art results as described in the Figure 12.

	Test Error (%)
Wan *et al.* (Wan et al., 2013)	0.28
Graham (Graham, 2014a)	0.31
Untied 3-LSTM	**0.32**
Ciresan *et al.* (Ciresan et al., 2012)	0.35
Untied 3-LSTM with ReLU (*)	**0.36**
Mairar *et al.* (Mairal et al., 2014)	0.39
Lee *et al.* (Lee et al., 2015)	0.39
Simard *et al.* (Simard et al., 2003)	0.4
Graham (Graham, 2014b)	0.44
Goodfellow *et al.* (Goodfellow et al., 2013)	0.45
Visin *et al.* (Visin et al., 2015)	0.45
Lin *et al.* (Lin et al., 2013)	0.47

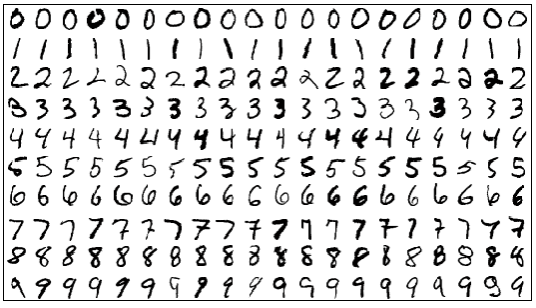

Figure 12,13: MNIST accuracy table where GridLSTM achieves near-state-of-the-art results (left), MNIST dataset preview sample set (right).

3.4.1 Dataset

Dataset consists from 50,000 training and 10,000 testing handwritten scanned images of size 28x28 pixels. In the experiment, data augmentation is applied by shifting the digits randomly 0-4 pixels in both vertical and horizontal direction. Data augmentation helps the model to generalize well on unseen images by extending artificially the training set. In the current mode we don't use validation set, which is usually used to prevent overfitting the model.

3.4.2 Model

For solving handwritten digit classification, 3-LSTM is constructed with 4 layers. The structure is similar to multidirectional RNN. However, instead of having parallel layers, they are connected in depth. The final layer concatenates all output (including memory cells) of the GridLSTM module into final ReLU layer with 4096 cells. Final Softmax layer is applied to the output. Each GridLSTM block contains 100 cells.

Module takes an input of 2x2 (or 4x4) non-overlapping patches of the image. Each patch is linearized and normalized by dividing it 256. Each layer is directional. In other words, the image is parsed from each corner on each layer as depicted in Figure 14. The final layer represents the classification of the digit.

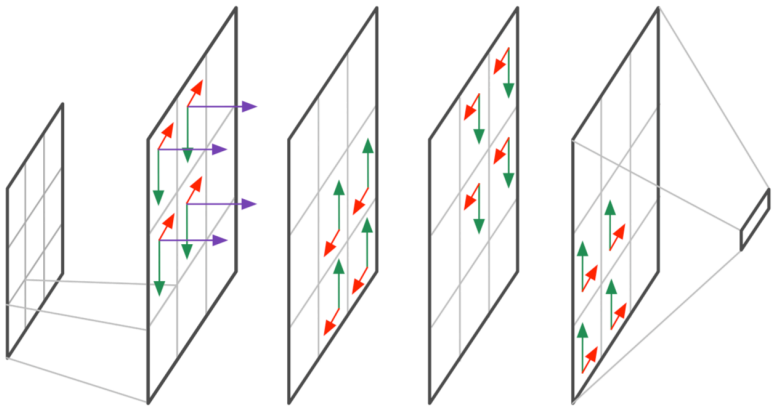

Figure 14: 4 Layer GridLSTM for Digit Classification with concatenation layer at the end

3.4.3 Training

For computing the loss, negative log likelihood criterion is used in conjunction with stochastic optimization Adam with learning rate 0.001. Batch size is 128 (The number of samples trained at once). The model was trained on Azure VM with 16 cores and 128GB RAM on CPU. The GPU implementation is also tested on Titan X GPU with 3072 CUDA cores.

3.4.4 Results

After training 24 hours on CPU or 2 hours GPU, our implementation reached 99.2% accuracy on 10,000 testin images. We have tried variations of the model including different patch size and rnn size for finding any possibility of slightly better architecture that could be compared to the one presented in the paper.

Figure 16, shows the comparison of GridLSTM and Convolutional Neural Network accuracies along 23 epochs (60 batch iterations). In terms of performance CNN slightly outperform GridLSTM initially however, in the long term they converge. The paper didn't precisely state if the 3-LSTM trained on MNIST had priority dimensions. As shown on Figure 16, priority dimensions do not add any effectiveness. Directional layer most probably causes indifference as the prediction encounters all contextual information including itself along depth axes.

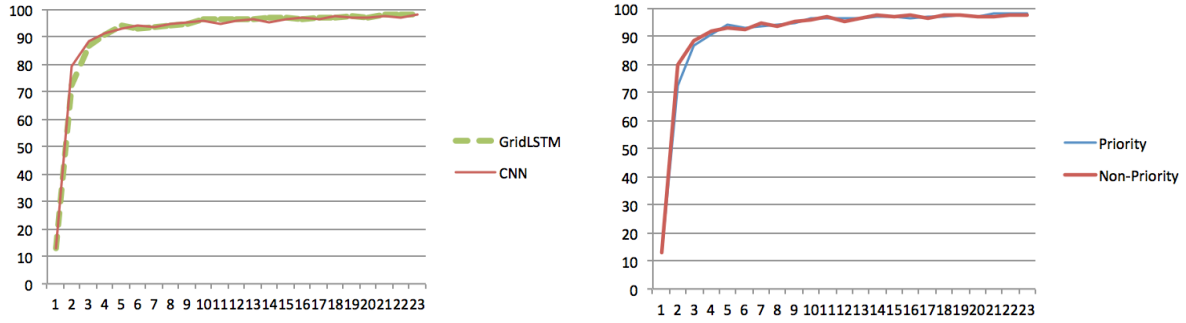

Figure 15,16: Grid LSTM and CNN comparison, GridLSTM with Priority Dimensions and Non-Priority Dimension.

3.5 Evaluation

It takes longer to train GridLSTM rather than Convolutional Neural Network. GridLSTM (4.3ms per sample) is 10 times slower than CNN (0.42ms per sample), while computing on GPU. GPU implementation of GridLSTM makes the algorithm 10 times faster as opposed to CNN where GPU boosts 16x times. This is based on the fact that CNN is natively implemented on GPU. Varying the size of patches linearly affects the computational time.

From the other perspective, based on the number of the clones, the model heavily demands memory. For example, for 200 clones (almost 14x14) with 128 batches the RAM resources required to run and train the algorithm is 16 GB. Based on the number of weight matrix stored and cloned across dimensions. Torch framework's hack of unfolding RNNs makes them limited for long sequences, however on the other hand, very fast for processing. In contrast, Theano (python based framework) gives the

possibility either to unfold the recurrence to process it instantly or keep the architecture feed forward sequentially. It gives the advantage for developers to decide the balance between computational power and memory consumption.

As our implementation is based on Lynch's one-dimensional implementation, we will discuss our contribution in terms of development and its advantages. Unlike Lynch's implementation, our implementation is modular. So, it could be plugged into any deep neural network by defining a GridLSTM within a single line as shown below.

```
nn.GridLSTM(height, width, input size, [parameters])
```

It is more optimized and at least 4x faster based on the recursive cloning of RNN package. Unlike directly processing depth dimension, our implementation only handles one layer such that user can define the number of sequential layers by adding nn.GridLSTM([...]) into the Sequential module. This adds more flexibility: for example, dynamically adding directional layers. In addition to that adding several loops in the model itself could easily result in N dimensional grid, which could be used for scanning MRI Scans.

Even though current state of the GridLSTM is resource demanding, the model could be applied in conjunction with several techniques to image processing tasks that will be discussed in the next chapter.

4 Deep Vision

4.1 Dataset

4.1.1 NYU

4.1.2 Superpixels

4.2 Semantic Segmentation

4.3 Depth Estimiation

5 Conclusion

5.1 Overview

5.2 Discussion

From theoretical point of view, one could notice a correlation between Integral Images and Recurrent Neural Networks. Integral Images have been introduced for efficient evaluation of the sums of image values aligned over rectangular region. Furthermore, it has been successful in real-time face detection and other computer vision tasks including depth estimation [17, 18]. RNNs in computer vision might have profound explanation of how spatiotemporal context affects efficiency and effectiveness of the model. For example, in the scene-labeling task, 2D LSTM achieved a top result with running time on a single Central Processing Unit (CPU) compared to other counterparts trained on Graphical Processing Unit (GPU) [13]. Thus thorough examination of the correlation will provide significant background for RNNs' further potential in low-level and high-level computer vision problems.

5.3 Future Work

Deep Vision achieved a new wave of state-of-the-art solutions for computer vision problems, mainly due to CNNs. However, they often lack accuracy in pixel-level labeling tasks. Proposed Recurrent Neural Network is going to consider spatiotemporal context of the pixel and estimate depth reconstruction with real-time processing in 'mind'. Based on the integral image concept and its usage, one might think of reversing CNN-RNN usual integration so that RNN produces integral image first and only then, on top, CNN will extract features. Andrej Karpathy called the unreasonable effectiveness of recurrent architectures magic, explained and visualized RNNs [21]. My goal during this research is to unveil this magic by providing profound explanation under deep vision context. Furthermore, using Depth Reconstruction as a specific use case, I shall introduce methods that could be applied to other computer vision problems by expanding the borders of Deep Vision and provide industry-ready solutions for robotics and other domains with real-time processing capabilities.

6 Bibliography

[1] Horry, Youichi, Ken-Ichi Anjyo, and Kiyoshi Arai. "Tour into the picture: using a spidery mesh interface to make animation from a single image." Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 1997.

[2] Szeliski, Richard, and Philip HS Torr. "Geometrically constrained structure from motion: Points on planes." 3D Structure from Multiple Images of Large-Scale Environments. Springer Berlin Heidelberg, 1998. 171-186.

[3] Zhang, Ruo, et al. "Shape-from-shading: a survey." Pattern Analysis and Machine Intelligence, IEEE Transactions on 21.8 (1999): 690-706.

[4] Hoiem, Derek, Alexei A. Efros, and Martial Hebert. "Automatic photo pop-up." ACM Transactions on Graphics (TOG) 24.3 (2005): 577-584.

[5] Delage, Erick, Honglak Lee, and Andrew Y. Ng. "A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image." Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. Vol. 2. IEEE, 2006.

[6] Saxena, Ashutosh, Sung H. Chung, and Andrew Y. Ng. "Learning depth from single monocular images." Advances in Neural Information Processing Systems. 2005.

[7] Saxena, Ashutosh, Min Sun, and Andrew Y. Ng. "Make3d: Learning 3d scene structure from a single still image." Pattern Analysis and Machine Intelligence, IEEE Transactions on 31.5 (2009): 824-840.

[8] Liu, Beyang, Stephen Gould, and Daphne Koller. "Single image depth estimation from predicted semantic labels." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[9] Konrad, J., et al. "Automatic 2d-to-3d image conversion using 3d examples from the internet." IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 2012.

[10] Konrad, Janusz, Meng Wang, and Prakash Ishwar. "2d-to-3d image conversion by learning depth from examples." Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on. IEEE, 2012.

[11] Ladicky, Lubor, Jianbo Shi, and Marc Pollefeys. "Pulling things out of perspective." Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014.

[12] Zhang, Guofeng, et al. "Consistent depth maps recovery from a video sequence." Pattern Analysis and Machine Intelligence, IEEE Transactions on 31.6 (2009): 974-988.

[13] Karsch, Kevin, Ce Liu, and Sing Bing Kang. "Depth extraction from video using non-parametric sampling." Computer Vision–ECCV 2012. Springer Berlin Heidelberg, 2012. 775-788.

[14] MLA Eigen, David, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network." Advances in Neural Information Processing Systems. 2014.

[15] Liu, Fayao, Chunhua Shen, and Guosheng Lin. "Deep convolutional neural fields for depth estimation from a single image." arXiv preprint arXiv:1411.6387 (2014).

[16] Konda, Kishore, and Roland Memisevic. "Unsupervised learning of depth and motion." arXiv preprint arXiv:1312.3429 (2013).

[17] Graves, Alex. Supervised sequence labelling with recurrent neural networks. Vol. 385. Heidelberg: Springer, 2012.

[18] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, 1997.

[19] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." Signal Processing, IEEE Transactions on 45.11 (1997): 2673-2681.

[20] Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." arXiv preprint arXiv:1412.2306 (2014).

[21] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell, Long-term Recurrent Convolutional Networks for Visual Recognition and Description, arXiv:1411.4389 / CVPR 2015

[22] Graves, Alex, Santiago Fernandez, and Ju3rgen Schmidhuber. "Advances in Neural Network Architectures-Multi-dimensional Recurrent Neural Networks." Lecture Notes in Computer Science 4668 (2007): 549-558.

[23] Byeon, Wonmin, et al. "Scene Labeling with LSTM Recurrent Neural Networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[24] van den Oord, Aaron; Kalchbrenner, Nal; Kavukcuoglu, Koray, "Pixel Recurrent Neural Networks", arXiv:1601.06759 (2016)

[25] Kalchbrenner, Nal, Ivo Danihelka, and Alex Graves. "Grid long short-term memory." arXiv preprint arXiv:1507.01526 (2015).

[26] Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhutdinov. "Unsupervised learning of video representations using lstms." arXiv preprint arXiv:1502.04681 (2015).

[27] Shi, Xingjian, et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." arXiv preprint arXiv:1506.04214 (2015).

7 Appendices

A Benchmarking

Experimentation

Datasets

Available depth datasets are available.

- Make3D - make3d.cs.cornell.edu/data.html
- B3DO - kinnectdata.com
- NYU depth v1 and v2 - cs.nyu.edu/˜silberman/datasets
- RGB-D dataset - cs.washington.edu/rgbd-dataset
- DepthTransfer - kevinkarsch.com/depthtransfer
- SUN3D - http://sun3d.cs.princeton.edu

Benchmarking

In the below, available benchmarking criteria for comparison

- Average relative error (rel): $\frac{1}{T}\sum_p \frac{\left|d_p^{gt}-d_p\right|}{d_p^{gt}}$
- Root mean squared error (rms): $\sqrt{\frac{1}{T}\sum_p\left(d_p^{gt}-d_p\right)^2}$
- Average $\log_{10}$ error (log10): $\frac{1}{T}\sum_p\left|\log_{10} d_p^{gt}-\log_{10} d_p\right|$
- Accuracy with threshold *thr*: percentage (%) of d_p s.t: $max(\frac{d_p^{gt}}{d_p},\frac{d_p}{d_p^{gt}}) = \delta < thr$;

Where d_p^{gt} and d_p are the ground-truth and predicted depths respectively at pixel indexed by p, and T it is the total number of pixels in all the evaluated images.

Current Results

State-of-the-art results on NYU v2 dataset

Method	Error (lower is better)			Accuracy (higher is better)		
	rel	log10	rms	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Make3D	0.349	-	1.214	0.447	0.745	0.897
DepthTransfer	0.35	0.131	1.2	-	-	-
Discrete-continuous CRF	0.335	0.127	1.06	-	-	-
Ladicky et al.	-	-	-	0.542	0.829	0.941
Multi-Scale Deep Network	**0.215**	-	0.907	0.61	**0.887**	**0.971**
Convolutional Neural Field	0.230	**0.095**	**0.0824**	**0.614**	0.883	**0.971**

State-of-the-art results on Make3D dataset

Method	Error (C1) (Lower is better)			Error (C2) (Higher is better)		
	rel	log10	rms	rel	log10	rms
Make3D	-	-	-	0.370	0.187	-
Semantic Labelling	-	-	-	0.379	0.148	-
DepthTransfer	0.355	0.127	9.20	0.361	0.148	15.10
Discrete-continuous CRF	0.335	0.137	9.49	0.338	0.134	13.29
Convolutional Neural Field	**0.314**	**0.119**	**0.860**	**0.307**	**0.125**	12.89

*These data is taken from [15] and not all models in the paper are presented