Recurrent Networks in Residual Networks

Poojita Thukral
Carnegie Mellon University
Forbes Avenue
pthukral@andrew.cmu.edu

Prachee Sharma
Carnegie Mellon University
Forbes Avenue
prachees@andrew.cmu.edu

Prithvishankar Srinivasan
Carnegie Mellon University
Forbes Avenue
prithvis@andrew.cmu.edu

Abstract

Deep learning is the upcoming field of research as it allows us to train deeper networks. The advantage of deeper networks is that it improves the classification efficiency, which is because as the number of layers increase, the network learns more about the underlying structure of the image, thus providing more informative feature maps. We follow research of Qianli Liao, Tomaso Poggio[1] who claim that a shallow RNN is exactly equivalent to a very deep ResNet with weight sharing among the layers. Our approach towards this project is to work towards Recurrent ResNets so that we can reproduce the same results as a deep ResNet by using a shallow RNN.

1. Introduction

Convolutional Neural Networks (CNNs) are inspired by biological networks that exist inside the brain. Just like how neurons are interconnected in subregions of the visual cortex, neurons inside a CNN are inter-connected to the subsequent hidden layer. These networks have multiple layers, starting from filters which are used for convolution with the input image. These filters weights are used to highlight features of the image, and they are shared within the network, resulting in reduction of the time required to learn these parameters. Operations like pooling ensure translational invariance in the image. As the number of layers are incremented, the network gains more information about the system, and it produces higher or more defined features. Use of these high-level features has been on a rise for the past few decades as they have wide applications in the field of classification. Also, CNNs are preferred over fully connected networks as they are relatively easier to train. It has been proved that efficiency of these networks can be increased by adding more layers to the network. But an underlying limitation is that as the hidden layers are increased, the vanishing gradient problems arises and the deep network poses bigger challenges when it comes to training.

Recently, Microsoft proposed a solution to this problem. They came up with a model called Residual Networks or ResNets which are essentially just a modification to CNNs. In ResNets, input from a layer is fed to one of subsequent layers so that the value of the gradient is not lost while traversing through deep networks. This also ensures that another layer is added only if it is more informative for the system. Thus, as ResNets overcome the main problem posed by CNNs, they can be used to train ultra-deep neural networks.

In the recent years, the state-of-the-art performance achieved by residual networks for classifying various data sets has resulted from ultra-deep architectures which have been shown to perform better than shallower networks consistently [1]. In [2] ,Qianli Liao and Poggio demonstrate that the basis of the good performance of such ultra-deep architectures is their efficiency in performing recurrent computations. It's shown that a deep residual network is equivalent to a shallow RNN. Further they demonstrate that an RNN with weight sharing (and therefore, less parameters) can yield performance similar to its corresponding ultra-deep residual network.

In this project, we first describe the research that has been done related to this field. We then talk about the approach towards our aim and the dataset and tools that we used for the implementation. This is followed by the architecture that we implemented and finally, we have stated the results that we observed.

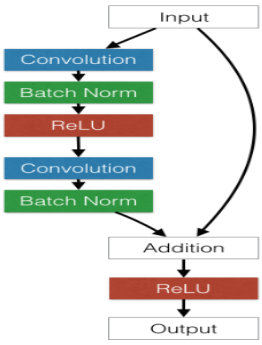

Fig 1. Basic ResNet Building Block

2. Related Work

Compared to normal CNNs, ResNets make the learning process faster by providing gradients a clearer path to back propagate to early layers of the network, thus avoiding the vanishing gradient problem or dead neurons. Zhang and Rendescribe a srutcute of ResNets with layers

ranging from 18 to 152 in [2]. They show that ResNets can be viewed as multiple basic blocks which are sequentially connected to each other apart from having shortcut connections parallel to each basic block that contribute to their outputs. Sam Gross and Michael Wilber[3] compare different basic blocks for the shortcut connection shown in Figure [1]. It demonstrates how adding a parametered layer after addition can reduce advantages of ResNets because they leave no fast way for gradients to back propagate anymore. Adding an un-parametered layer like ReLu however, does not present any major advantage or disadvantage.

3. Approach

To implement a recurrent residual network it is imperative that we start from the basics for data set classification. Hence, after implementing a convolutional neural network without residual connections, we modified the structure to a residual network architecture by varying the locations of the various blocks in a convolutional neural network such as ReLU (rectified linear unit), batch normalisation etc. The input from previous convolution layer is fed to the subsequent convolution layer, and it is denoted by $F(x) + x$ operation. After analyzing the optimum network architecture for the residual network, we implemented recurrent models for residual networks. The equivalence of ResNet and their corresponding RecNet for can be seen in Figure 2.

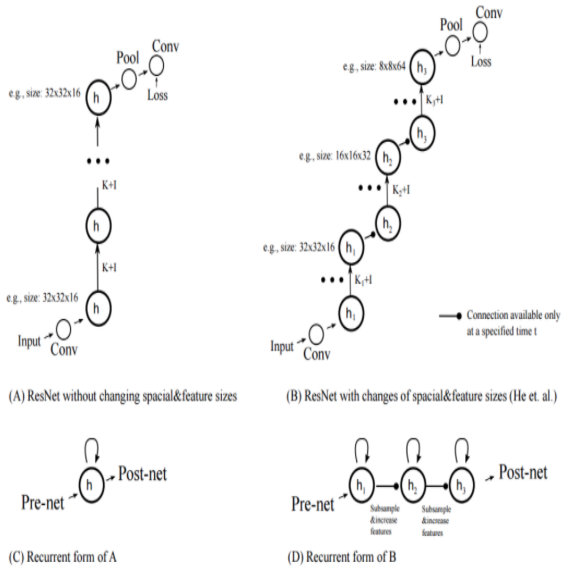

Fig 2. Recurrent form of networks

3.1. Network Architecture

Now we'll discribe the network architecture implemented for going from CNNs to ResNets, followed by the architecture of the final RecNet.

3.11 Residual Network from CNNs

A convolutional neural network was constructed to exploit the spatial relation between the different dimensions in the picture. For this purpose, we made use of MNIST dataset which consists of numbers ranges 0-9. 10,000 images were taken as training data and 1,000 images were taken as testing data. The input images were all of the dimension 28x28. For the convolutional layer, 20 filters of size 5x5 were chosen with a stride of 2.

Next, a pooling layer is used to condense the feature maps found in the output of the convolutional layer. This gives the output obtained from the convolutional layer in a dense form before feeding to the next convoluted layer. After the pooling layer, another set of convolutional and pooling layers were used to get higher level features. Then, an activation function, namely ReLU(Rectified Linear Unit) layer is used which which approximates to the analytic function,

$$f(x) = ln(1 + e^x)$$

Finally, a fully connected layer is used to classify the output using a softmax regression layer to classify outputs into 10 labels using one hot encoded format.

3.2 Recurrent Network Architecture

The 18-layer Resnet model introduced in [3] is a good start to understand the Resnet model. It uses different weights and not shared weights as the number of feature and spatial sizes change for every section of layers. A shortcut connection is added to each pair of 3x3 filters.

The results discussed in [4] suggest that this model is prone to overfit. Also, an 18 layer residual network is computationally intensive to train on CPUs.

4. Dataset and Implementation Tools

Here we describe the dataset and framework we used for training and network implementation.

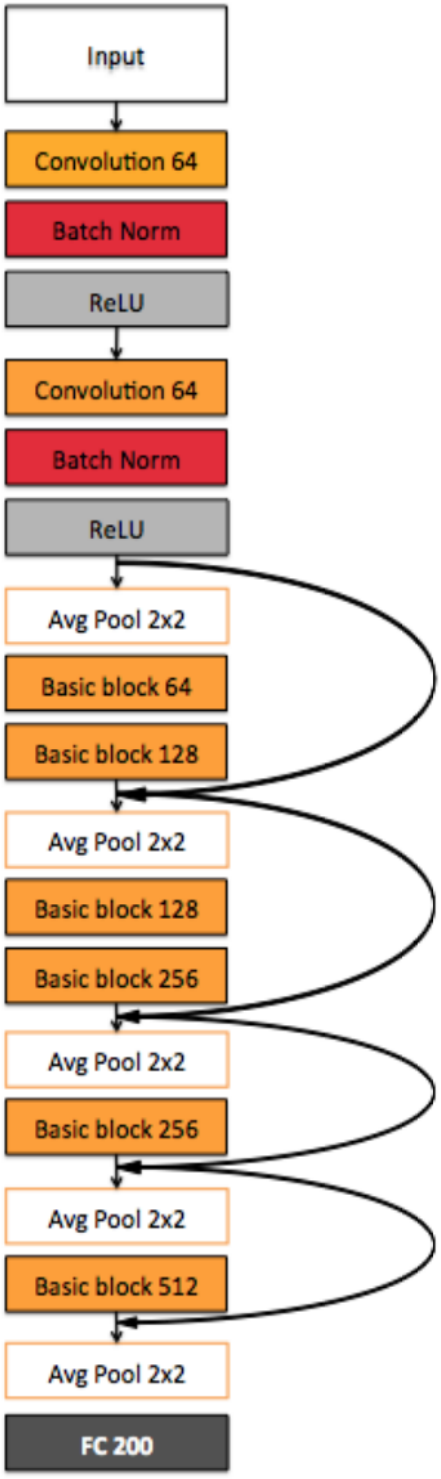

Figure 3 18-layers ResNet model

4.1 Dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. It consists of classes like airplane, bird, boats, automobile, train etc.

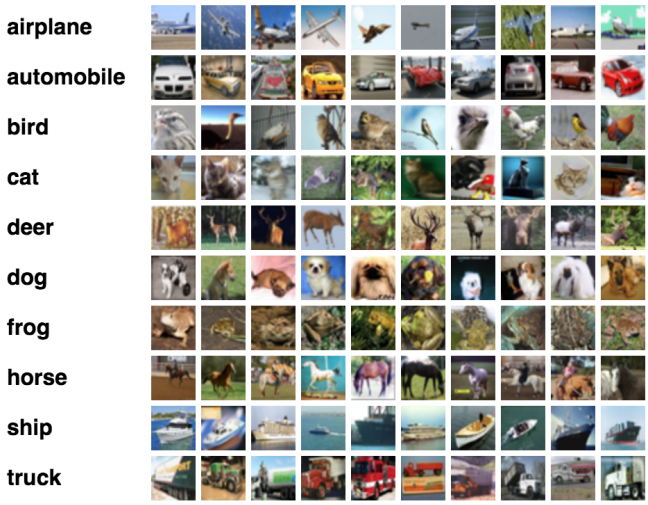

Fig 4. Sample images from CIFAR-10 dataset

4.2 Keras

Keras is an open source, high-level neural networks library, written in Python. It is capable of running on top of either TensorFlow or Theano[5].

Keras enables writing a deep learning library that is modular. A model in Keras code can be seen as a sequence of independent, configurable models which can be plugged together. For example, cost functions, regularization are all standalone modules. These blocks can be plugged together with minimal restrictions which enables easy extensibility of the library.

Keras supports convolutional networks, recurrent networks as well as a combination of the two. It has support for running on both CPU and GPU. In this project a xyz platform was used to implement and train different ResNet and ConvNet models as well as the Recurrent ResNet model.

4.3 Tensorflow

Tensorflow[6] works as a backend for Keras library. It aids in numerical computation using data flow graphs and the edges of the graph are represented in the form of tensors.

5. Experiments and Observation

Recurrency was incorporated in the model described in 5.1 by feeding flattened input to an RNN block. The architecture of the resultant RecNet was inspired from the figure shown in Figure 3. We unrolled our recurrent model seven times to add depth to the network.

As described in the picture, blocks coloured in blue are common to both shallow residual network and shallow recurrent network architecture. The depth of the residual network is just 1. Thus, it fails to converge and gives high misclassification error in both test and training data sets (approx 54%).

For the recurrent network implementation, a simple RNN block is added 7 times. This is equivalent to having seven residual network blocks sequentially lined after the first one. The testing error did improve and approximately resulted in 45% misclassification error. However, we seemed to realise that due to the shallow nature of the architecture, the training data seemed to be overfitted. Shown in figures 5 & 6 are misclassification errors vs number of epochs. Both architectures were trained for over 30 epochs.

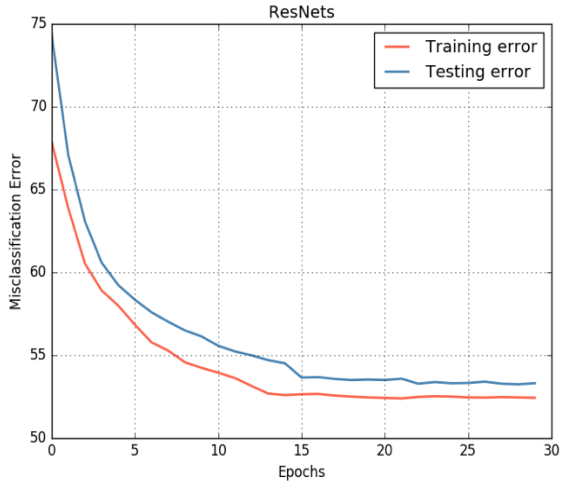

Fig 5. Misclassification error vs Epochs plot for deep ResNet architecture

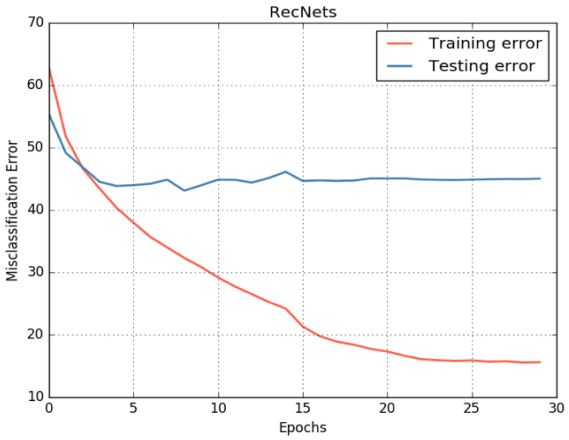

Fig 6. Misclassification error vs Epochs plot for RecNet architecture

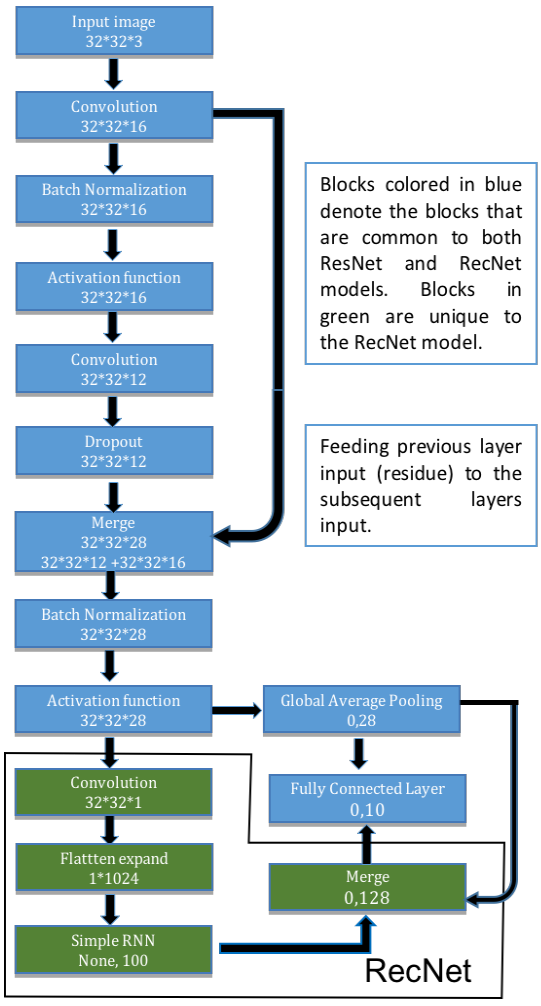

Fig 7. Implemented ResNet and RecNet architecture.

7. Future Work

Low accuracy achieved by our ResNet model suggests that we need to incorporate more deep blocks for our system to train better. For RecNets we need to modify our system architecture for better simulation results. Changes can be made with respect to how we train our models or the activation functions in order to make RecNets replicate ResNets successfully. Some of the potential techniques can be removing the dropout layer or changing the number of times the Recurrent layer is unrolled.

8. References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. arXiv preprint arXiv:1603.05027, 2016

[2] Qianli Liao, Tomaso Poggio, Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex, Center for Brains, Minds and Machines, McGovern Institute, MIT,CBMM Memo No. 047

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition.1512.03385, 2015.

[4] http://cs231n.stanford.edu/reports2016/264_Report.pdf

[5] https://keras.io/

[6] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

[7] http://torch.ch/blog/2016/02/04/ResNets.html, Sam Gross and Michael Wilber

[8] LeCun, Yann; Corinna Cortes; Christopher J.C. Burges. "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges". Retrieved 13 November 2016.

[9] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[10] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. Understanding deep architectures using a recursive convolutional network. arXiv preprint arXiv:1312.1847, 2013.