International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 10 | Oct 2019 www.irjet.net p-ISSN: 2395-0072

Spatial Context Preservation and Propagation - Layer States in Convolutional Neural Networks

Susmit Agrawal[1]

[1]Student, Department of Computer Science and Engineering, Sir M. Visvesvaraya Institute of Technology, Bangalore

---***---

Abstract - *Recurrent Neural Networks use hidden states to preserve information in sequential data. Loss of context is a problem also seen in Deep Convolutional Neural Networks, for example in object detection when the size of the object is much smaller than the receptive field at the final layer. Architectures such as ResNet and Inception have proven to be quite effective to solve the issue, but even they cannot preserve context over several layers. We propose a novel approach to propagate context over layers in CNNs, where layers of the network produce context vectors as a function of their outputs, and each context vector depends on the context vector of previous layers. The function producing the context vector may be a miniature neural network in itself. This approach can be applied as an add-on to existing architectures, such as VGG networks or ResNets, without requiring any change in the base model. We compare a state-augmented model to the base model and to ResNet18, and find that it considerably outperforms both in classification tasks.*

***Key Words*: Convolutional Neural Network, State Vector, Context Vector, Computer Vision, Image Processing**

1. INTRODUCTION

Preserving image context in deeper layers of Convolutional Neural Networks has become an age old problem in the field of deep learning. In its most basic form, the problem is that in a standard CNN, a layer depends directly on its previous layer to provide all required features for further processing. Any features lost by a layer during the correlation operation are lost to the remaining network. This problem is further escalated by operations used for reducing dimension sizes, such as pooling and 1x1 convolutions.

All CNN architectures present today attempt to encode both features and processed context into a single multidimensional vector, to an extent where the latter cannot be distinguished from the former.

1.1 Understanding Residual Connections in modern architectures

Many variants of Convolutional Neural Networks use the concept of Residual Connections or Skip Connections. The idea of residual connections has allowed development of very deep networks, some having over a hundred layers.

A residual connection, in its most basic form, is the output of a layer that is added to the output of another layer deeper in the network. Intuitively, residual connections allow image context to be forwarded directly to a deeper region of the network. However, any features extracted by the layer are also forwarded. This may lead to corruption of features that have been extracted by the deeper layer.

1.2 Introducing Layer States

In this work, we attempt to isolate context encoded within a convolutional layer's output. This context is preserved separately and combined with a digest of contexts extracted from previous layers. We call this context the **State** of the layer.

To put this in a mathematical perspective, assume that a single block of a convolutional neural network performs the following operation on a given input, *X*:

$$CF = f(X) \tag{1}$$

Here, *CF* is a single vector containing both context and features. We assume that any other parameters, such as weights or biases, are defined or initialized inside *f*.

The very next block would then perform another computation as follows:

$$C'F' = f(CF) \tag{2}$$

It is seen here that the context of the previous layer is modified and new features are extracted simultaneously from the same vector.

This work tries isolating the context from the combined vector. Intuitively, we attempt to extract the context and process it separately as follows:

$$C = g(CF, C_{prev}) \tag{3}$$

C is termed the Layer State.

The general form of the above expression for any layer *n* would be:

$$C_n = g((CF)_n, C_{n-1}) \tag{4}$$

where C_n is the Layer State for layer *n*.

The second term of the function is the context vector from the previous layer. The initial value, C_0, may be initialized manually or learnt during training.

When the flow reaches the final convolutional block, C_{fin}, context is computed as:

$$C_{fin} = g((CF)_{fin}, C_{fin-1}) \quad (5)$$

The model's prediction can then be computed as a function of C_{fin}:

$$Y = h(C_{fin}) \quad (6)$$

2. IMPLEMENTATION ON TOP OF AN EXISTING ARCHITECTURE

One key feature of this approach is that it does not require major changes in the structure of the underlying network. We demonstrate the concept by adding states to a basic CNN architecture, and comparing the results with the base model and with the ResNet18 architecture. We compare the models on a classification task, using CIFAR10 dataset, CIFAR100 dataset with fine labels, and Tiny ImageNet.

2.1 The base architecture

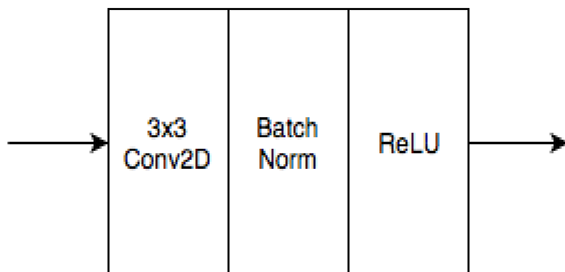

Fig - 1: A Convolutional Block

The base architecture is derived from the VGG16 architecture. An overview is shown in [Fig – 3]. It consists of Convolutional Blocks, shown in [Fig - 1], and MaxPooling layers, followed by a single Fully-Connected layer with Softmax activation at the end. Multiple blocks are stacked sequentially to create the network. All Conv2D layers use "same" padding – the height and width of input and output images is the same.

This network presents a Convolutional Neural Network in its most basic form. We use this model as a baseline for evaluating the performance of different architectures.

2.2 The modified architecture

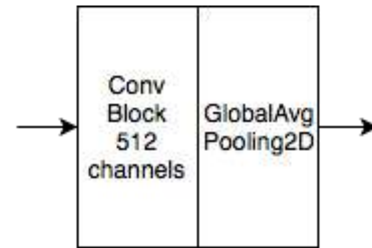

Fig - 2: A State Block

The modified version of the network, shown in [Fig - 4], consists of the base model augmented with State Blocks branched out before each MaxPooling layer. Structure of a State Block is shown in [Fig - 2]. Each State Block calculates the context vector for the block from which it receives its input. A Fully Connected layer with Softmax Activation is used at the end of the final State Block for classification.

The values of the initial state vector C_0 are set to zeros.

Here, the function *g* mentioned in equation (3) is defined as:

$$g(X_1, X_2) = X_1 + X_2 \quad (7)$$

2.3 Experimental Setup

- The networks were created and trained using Google's TensorFlow Framework.
- Spatial Dropout was used in order to prevent networks from overfitting. Without it, all networks began overfitting within the first few epochs.
- Training data consisted of fifty thousand images in case of CIFAR datasets, and hundred thousand images in case of Tiny ImageNet. No image augmentation was used.
- Networks were trained and evaluated on each individual dataset in separate runs.
- Stochastic Gradient Descent with Nesterov Momentum was used as the optimizer.
- All hyperparameters were kept constant while training the networks. All networks were trained with the same values for all hyperparameters.
- No other performance-enhancement techniques were used, except the ones mentioned above.

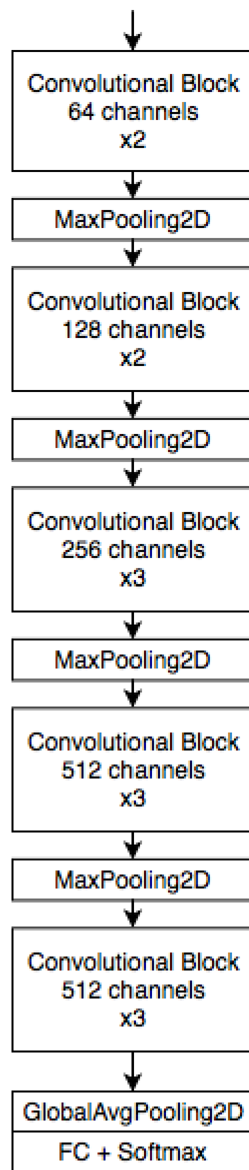

Fig – 3: Base CNN architecture

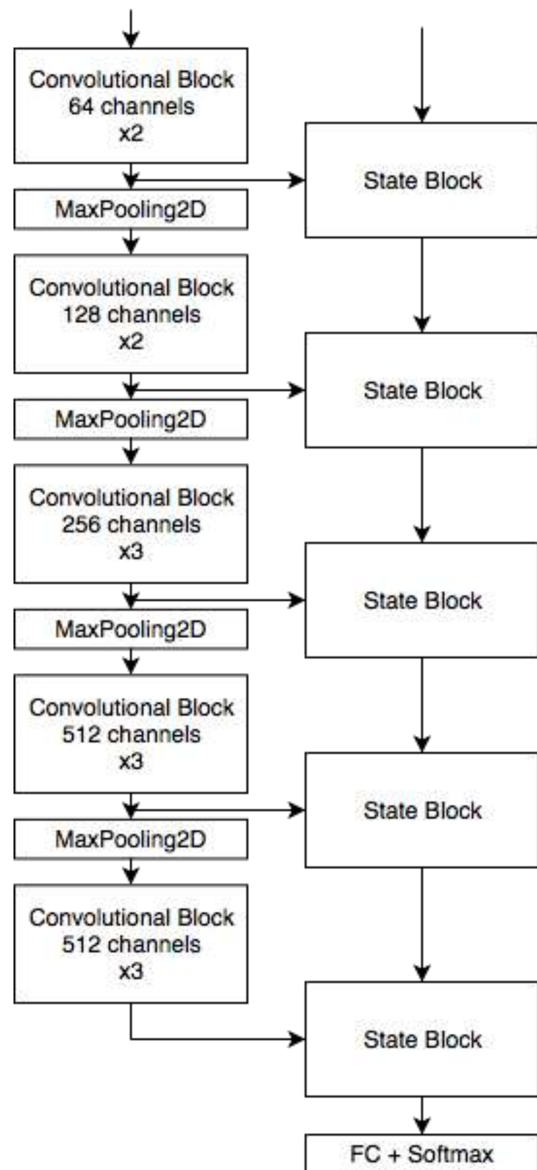

Fig – 4: Modified CNN architecture

3. CONCLUSIONS

The results of the experiments are plotted as graphs in figures [Fig – 5] to [Fig – 10].

The base architecture gives an expected, near-ideal curve for both loss and accuracy in all datasets.

From the graphs, we can infer that the ResNet architecture performs better in the presence of a large amount of complex data. It case of a simple dataset like CIFAR10, it is outperformed by a basic CNN, as it tends to overfit the training data.

The problem of overfitting in case of ResNet extends even to large, highly complex datasets. This is apparent by observing the upward curve in its loss functions after a number of epochs, as well as the fact that its validation accuracies tend to flatten out.

The impact of states on performance of a simple network is clear from the graphs. The accuracy of the combination on Tiny ImageNet is approximately 33% greater than the base architecture! When a relatively simple architecture based on VGG16 is augmented with State Blocks, it considerably outperforms a more complex model, the ResNet-18, on every task it is measured on.

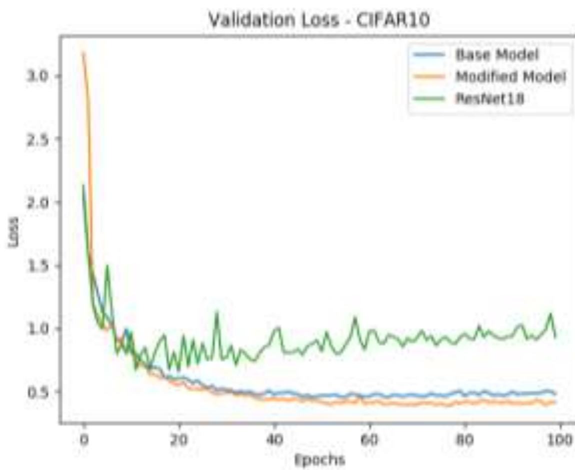

Fig - 5: Loss Comparison for CIFAR10

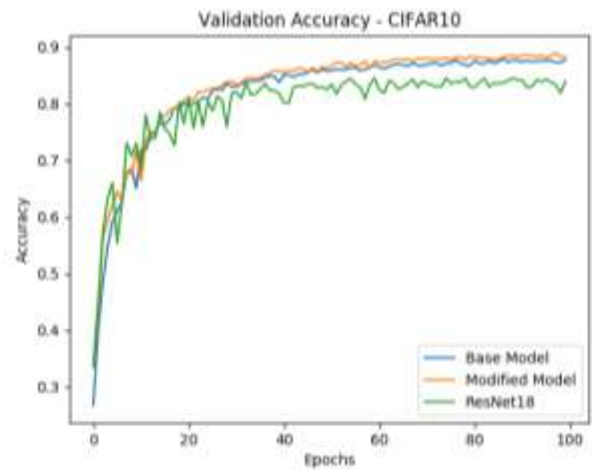

Fig - 6: Accuracy Comparison for CIFAR10

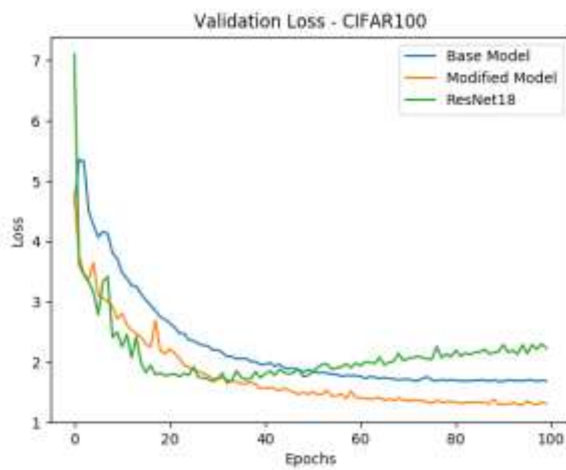

Fig - 7: Loss Comparison for CIFAR100

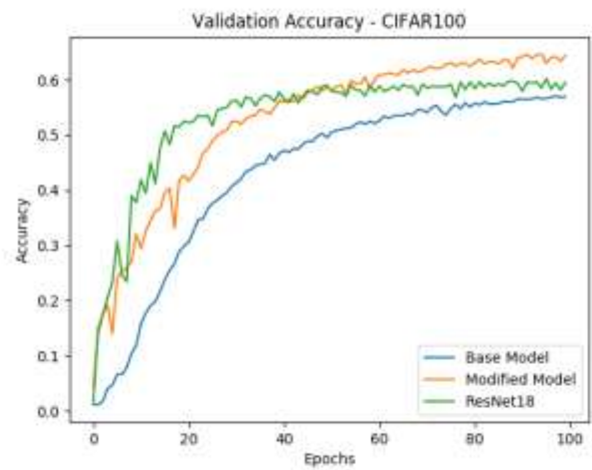

Fig - 8: Accuracy Comparison for CIFAR100

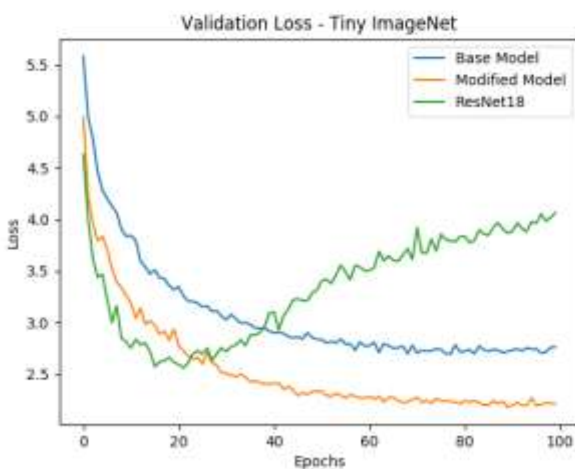

Fig - 9: Loss Comparison for Tiny ImageNet

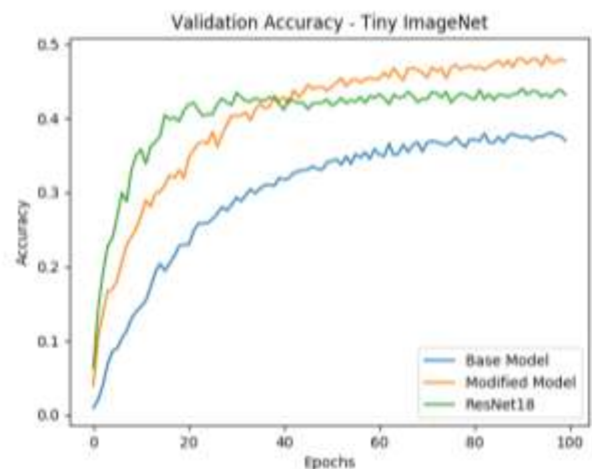

Fig - 10: Accuracy Comparison for Tiny ImageNet

REFERENCES

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun - Deep Residual Learning for Image Recognition

[3] Karen Simonyan, Andrew Zisserman - Very Deep Convolutional Networks for Large-Scale Image Recognition

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger - Densely Connected Convolutional Networks

[5] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, Huamin Qu - Understanding Hidden Memories of Recurrent Neural Networks

[6] Alex Krizhevsky - Learning Multiple Layers of Features from Tiny Images

[7] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu, Squeeze-and-Excitation Networks

[8] Zoheb Abai, Nishad Rajmalwar, DenseNet Models for Tiny ImageNet Classification

[9] Tee Connie, Mundher Al-Shabi, and Michael Goh, Smart Content Recognition from Images Using a Mixture of Convolutional Neural Networks

[10] Mohammad Sadegh, Hossein Karkeh Abadi, Study of Residual Networks for Image Recognition

[11] Rupesh Kumar Srivastava, Klaus Greff, Jurgen Schmidhuber, Highway Networks