Implementing Gradually Updated Neural Networks for Large-Scale Image Recognition

Aishwarya Radhakrishnan

May 5, 2020

1 Introduction

Convolutional Neural Networks(CNNs) decreases the computation cost for Computer Vision problems than the Deep Neural Networks(DNNs). Increasing the depth leads to increase in parameters of the CNN and hence, increase in computation cost. But as depth plays a vital role in Computer Vision problems, increasing the depth without increasing the computation cost can lead to increased learning. This is achieved by introducing computation orderings to the channels within convolutional layers. Gradually Updated Neural Networks (GUNN) as opposed to the default Simultaneously Updated Convolutional Network (SUNN / CNN), gradually updates the feature representations against the traditional convolutional network that computes its output simultaneously. This is achieved by updating one channel at a time and using the newly computed parameter of this channel and old parameter of other channels to get another channels in a single convolutional layer. This is repeated for all the channels untill all the old parameters of a single convolutional layer are updated to new values. Thus a single convolutional layer is broken down into multiple gradually updated convolutional layer.

2 Data

We do not need to download CIFAR-10 dataset explicitly as we Tensorflow provides cifar10.load_data() API which automatically downloads and loads Training and Test set in numpy ndarray. CIFAR-10 dataset has 32x32 dimensional images having objects from 10 classes as shown in fig. 1.

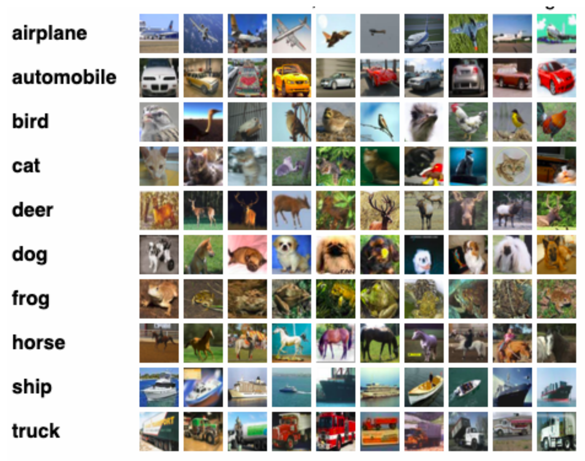


Figure 1: CIFAR 10 dataset with 10 classes that are randomly chosen

3 Procedure

1. As the original authors have used CIFAR-10 datasets, this project is focus on these datasets.

2. Used Google's Colab virtual environment for project implementation.

3. Used TensorFlow and Keras for implementing SUNN for CIFAR-10 dataset.

4. Converted SUNN to GUNN for CIFAR-10 dataset using Keras custom layer implementation.

5. Run simulations with different dataset subset.

6. Analyze the results.

4 Model

3.1 GUNN Layer implementation (Keras Custom Layer)

Keras Custom Layer needs to be used because GUNN layer consist of convolutional network which decomposes the original computation into multiple sequential channel-wise convolutional operations as opposed to single convolutional operations in Conv2D.

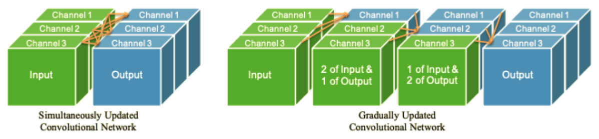
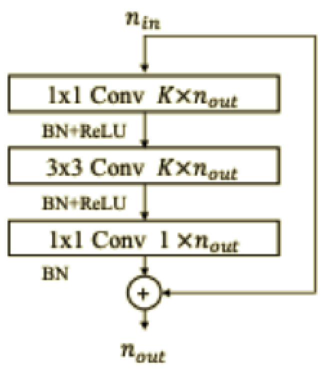


Figure 2: SUNN vs GUNN. SUNN architecture is our usually used Convolution Layer while GUNN is the channel-wise decomposed architecture.

The expansion rate is used to know how many channels are simultaneously updated when decomposing the channels. The kernel channel is the same as the expansion rate. The kernel size is given by fxf.


n_{in}
1x1 Conv $K\times n_{out}$
BN+ReLU
3x3 Conv $K\times n_{out}$
BN+ReLU
1x1 Conv $1\times n_{out}$
BN
n_{out}

Notation:

fxf : Kernel size
K: expansion rate
n_in: input channels
n_out: output channels

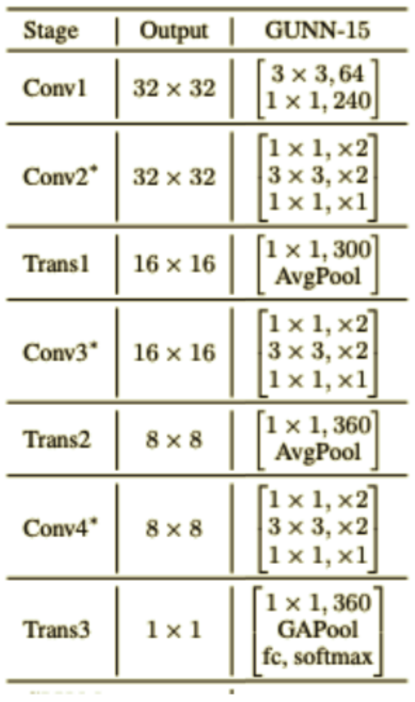
Figure 3: Gunn2D Keras custom layer. The Gunn2D layer expands channel at K rate and also uses Residual Network identity block implementation.

The Keras custom layer needs forward propagation definition only. Since our custom layer has trainable weights, we need to use stateful custom operations using custom layer class definition.

Note: If you want to build a keras model with a custom layer that performs a custom operation and has a custom gradient, you should use @tf.custom_gradient.

5 Building GUNN-15 Model in Keras for 10 classes of CIFAR-10 dataset

Using Keras Convolutional Neural Networks building blocks and custom implemented Gunn2D layer created GUNN-15 Model.



Stage	Output	GUNN-15
Conv1	32×32	$\begin{bmatrix} 3 \times 3, 64 \\ 1 \times 1, 240 \end{bmatrix}$
Conv2*	32×32	$\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$
Trans1	16×16	$\begin{bmatrix} 1 \times 1, 300 \\ \text{AvgPool} \end{bmatrix}$
Conv3*	16×16	$\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$
Trans2	8×8	$\begin{bmatrix} 1 \times 1, 360 \\ \text{AvgPool} \end{bmatrix}$
Conv4*	8×8	$\begin{bmatrix} 1 \times 1, \times 2 \\ 3 \times 3, \times 2 \\ 1 \times 1, \times 1 \end{bmatrix}$
Trans3	1×1	$\begin{bmatrix} 1 \times 1, 360 \\ \text{GAPool} \\ \text{fc, softmax} \end{bmatrix}$

Figure 4: Gunn Model. The Gunn Model has Gunn2D layer implemented at the layers marked as * i.e. Conv2*, Conv3* and Conv4* layers.

Built Gradually updated Convolutional Neural Net following GUNN-15 model having 15 layers that classifies CIFAR-10's 3 classes in Colab with the following architecture:

- Training examples : 5000
- Testing examples : 100
- Batch Size : 50
- epochs : 100
- Adam optimizer : for adaptive estimates of lower-order moments
- Loss function : Categorical crossentropy for multiple class estimation
- Activation layer : RELU
- Batch Normalization : for the network to use higher learning rate without vanishing or exploding gradients.

In addition, all the layers parameters are defined in accordance with the model in fig. 4.

I have also used Residual Network's Identity block for each Gunn2D layer where the state of layer before doing Gunn2D operations is added to the layer after performing Gunn2D operations. Residual network helps in learning identity function when the gradients vanishes which happened in deep networks.

The fig. 5 shows the High-Level Keras Automatic Differentiation graph having different layers like Conv2d, Activation etc that are Keras APIs as well as the **Gunn2D layer** which is a custom layer.

The parameters of the model are given in fig. 6

6 Training

Trained the Convolutional Neural Net to classify CIFAR-10's 3 classes in Colab using 5000 training examples and 100 testing examples with batch size of 50 and for 50 epochs, I get the following output on the test set.

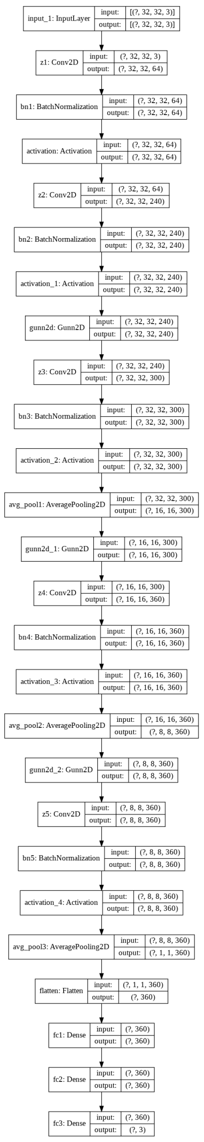
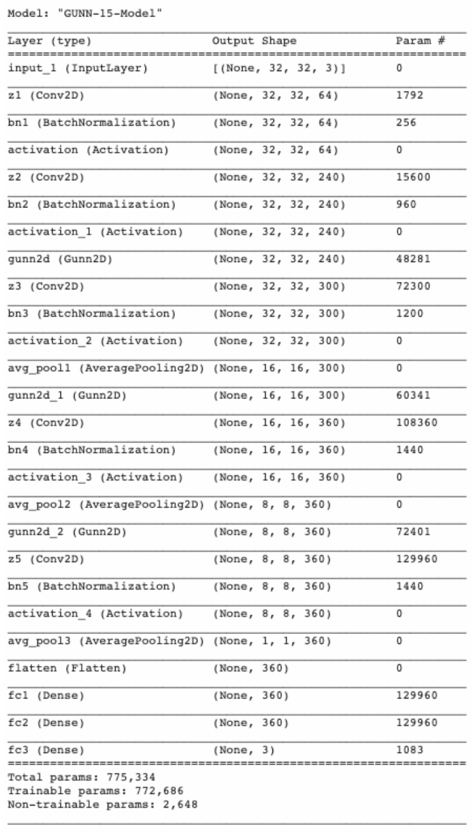

input_1: InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
z1: Conv2D | input: (?, 32, 32, 3) | output: (?, 32, 32, 64)
bn1: BatchNormalization | input: (?, 32, 32, 64) | output: (?, 32, 32, 64)
activation: Activation | input: (?, 32, 32, 64) | output: (?, 32, 32, 64)
z2: Conv2D | input: (?, 32, 32, 64) | output: (?, 32, 32, 240)
bn2: BatchNormalization | input: (?, 32, 32, 240) | output: (?, 32, 32, 240)
activation_1: Activation | input: (?, 32, 32, 240) | output: (?, 32, 32, 240)
gunn2d: Gunn2D | input: (?, 32, 32, 240) | output: (?, 32, 32, 240)
z3: Conv2D | input: (?, 32, 32, 240) | output: (?, 32, 32, 300)
bn3: BatchNormalization | input: (?, 32, 32, 300) | output: (?, 32, 32, 300)
activation_2: Activation | input: (?, 32, 32, 300) | output: (?, 32, 32, 300)
avg_pool1: AveragePooling2D | input: (?, 32, 32, 300) | output: (?, 16, 16, 300)
gunn2d_1: Gunn2D | input: (?, 16, 16, 300) | output: (?, 16, 16, 300)
z4: Conv2D | input: (?, 16, 16, 300) | output: (?, 16, 16, 360)
bn4: BatchNormalization | input: (?, 16, 16, 360) | output: (?, 16, 16, 360)
activation_3: Activation | input: (?, 16, 16, 360) | output: (?, 16, 16, 360)
avg_pool2: AveragePooling2D | input: (?, 16, 16, 360) | output: (?, 8, 8, 360)
gunn2d_2: Gunn2D | input: (?, 8, 8, 360) | output: (?, 8, 8, 360)
z5: Conv2D | input: (?, 8, 8, 360) | output: (?, 8, 8, 360)
bn5: BatchNormalization | input: (?, 8, 8, 360) | output: (?, 8, 8, 360)
activation_4: Activation | input: (?, 8, 8, 360) | output: (?, 8, 8, 360)
avg_pool3: AveragePooling2D | input: (?, 8, 8, 360) | output: (?, 1, 1, 360)
flatten: Flatten | input: (?, 1, 1, 360) | output: (?, 360)
fc1: Dense | input: (?, 360) | output: (?, 360)
fc2: Dense | input: (?, 360) | output: (?, 360)
fc3: Dense | input: (?, 360) | output: (?, 3)

Figure 5: GUNN-15 model having custon Gunn2D layer.

Loss = 0.87605534954071045
Test Accuracy = 0.6960

On the training set the accuracy achieved at 100th epoch was :

Loss = 0.8460
Test Accuracy = 0.7290



```
Model: "GUNN-15-Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
z1 (Conv2D)                  (None, 32, 32, 64)        1792
_________________________________________________________________
bn1 (BatchNormalization)     (None, 32, 32, 64)        256
_________________________________________________________________
activation (Activation)      (None, 32, 32, 64)        0
_________________________________________________________________
z2 (Conv2D)                  (None, 32, 32, 240)       15600
_________________________________________________________________
bn2 (BatchNormalization)     (None, 32, 32, 240)       960
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 240)       0
_________________________________________________________________
gunn2d (Gunn2D)              (None, 32, 32, 240)       48281
_________________________________________________________________
z3 (Conv2D)                  (None, 32, 32, 300)       72300
_________________________________________________________________
bn3 (BatchNormalization)     (None, 32, 32, 300)       1200
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 300)       0
_________________________________________________________________
avg_pool1 (AveragePooling2D) (None, 16, 16, 300)       0
_________________________________________________________________
gunn2d_1 (Gunn2D)            (None, 16, 16, 300)       60341
_________________________________________________________________
z4 (Conv2D)                  (None, 16, 16, 360)       108360
_________________________________________________________________
bn4 (BatchNormalization)     (None, 16, 16, 360)       1440
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 360)       0
_________________________________________________________________
avg_pool2 (AveragePooling2D) (None, 8, 8, 360)         0
_________________________________________________________________
gunn2d_2 (Gunn2D)            (None, 8, 8, 360)         72401
_________________________________________________________________
z5 (Conv2D)                  (None, 8, 8, 360)         129960
_________________________________________________________________
bn5 (BatchNormalization)     (None, 8, 8, 360)         1440
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 360)         0
_________________________________________________________________
avg_pool3 (AveragePooling2D) (None, 1, 1, 360)         0
_________________________________________________________________
flatten (Flatten)            (None, 360)               0
_________________________________________________________________
fc1 (Dense)                  (None, 360)               129960
_________________________________________________________________
fc2 (Dense)                  (None, 360)               129960
_________________________________________________________________
fc3 (Dense)                  (None, 3)                 1083
=================================================================
Total params: 775,334
Trainable params: 772,686
Non-trainable params: 2,648
_________________________________________________________________
```

Figure 6: GUNN-15 model training parameters breakdown for all the stateful layers.

The best training set accuracy which was achieved at 91th epoch was :

Loss = 0.8430
Test Accuracy = 0.7620

Note: Require to train the whole classifier at once as checkpointing of this model is not possible. This is because the weights in our custom Keras layer are changed multiple times using multiple operations in a single batch step. Hence, saving of such a weight tensor is not yet defined in Tensorflow.

7 Evaluation

The GUNN-15 model in the original paper achieved 80% accuracy for 10 class classification. In the above model we have achieved 69.60% accuracy on the test set after training on training and label subset of CIFAR-10 dataset for 3 classes. The best training accuracy achieved was 76.20% hence early stopping also could have helped.

8 Results / Conclusion

Thus, we can see that convolutional neural networks when expanded depth-wise without increasing number of weights causes good classifier of images and also does not increase the computational cost. A single convolutional layer in CNN when broken down into multiple gradually updated convolutional layer would increases the depth and learning capability without increasing the computation cost. Also, the residual network properties are seen as for such a deep network the gradients have not vanished or exploded.

9 References

[1] Qiao, Siyuan et al. “Gradually Updated Neural Networks for Large-Scale Image Recognition.” ICML (2017).

[2] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), Advances in Neural Information Processing Systems, pp. 1097–1105. 2012.

[3] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. IEEE Conference on Computer Vision and Pattern Recognition, 2016a.

[4] He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. ECCV, 2016b.

[5] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning, 2015.