*International Journal of Engineering & Technology*, 7 (3.31) (2018) 66-71

International Journal of Engineering & Technology

*Website: www.sciencepubco.com/index.php/IJET*

***Research paper***

# Effective and High Computing Algorithms for Convolution Neural Networks

**P.Syamala Rao [1], Dr. G.P.SaradhiVarma [2], Rajasekhar Mutukuri [2]**

*[1] Department of Computer Science, Acharya Nagarjuna University, Guntur,India.- 522002.*
*[2] Department of Information technology,SRKR Engineering college, Bhimavaram India.-534204.*
**Corresponding author E-mail: peketi.shyam@gmail.com*

**Abstract**

Training a large set of data takes GPU days using Deep convolution neural networks which are a time taking process. Self-driving cars require very low latency for pedestrian detection. Image recognition constrained by limited processing resources for mobile phones. The computation speed of the training set determines that in these situations convolution neural networks was a success. For large filters, Conventional Faster Fourier Transform based convolution is preferably fast, yet in case of small, 3 × 3 filters state of the art convolutional neural networks is used. By using Winograd's minimal filtering algorithms the new class of fast algorithms for convolutional neural networks was introduced by us. Instead of small tiles, minimal complexity convolution was computed by the algorithms, this increases the computing speed with small batch sizes and small filters. With the VGG network, we benchmark a GPU implementation of our algorithm and at batch sizes from 1 to 64 state of the art throughput was shown.

***Keywords***: *Deep convolution neural networks, fast convolution neural network Algorithm, CGEM Methodology*

## 1. Introduction

Image recognition in state of the art results [1,2] is acquired by using deep convolution neural networks (convnets). Several days of GPU time is taken for training in these networks and it requires significant compute resources during classification too. Better accuracy can obtain from bigger data sets and models but it leads to increase in computation time. In deep neural networks, the speed of computation of the networks will determine the progress.

Likewise, when convnets are applied on low latency inference problems, such as to determine the how fast is tiny set of image data that can be determined and classified will limits the detection of people detection in autonomous cars in a video imagery.

By partitioning each group of samples across the nodes of a cluster. Distributed training of convnets(convolution neural networks) can be acquired and accumulating weight updates across the nodes. Convergence of the network can be affected by large batch sizes adversely, so the upper limit on the cluster size was placed with the minimum group size can be computed efficiently.

3 × 3 convolutional layers in deep networks are used for image recognition in state of the art convent(convolution neural networks) architectures, as fewer weights give better accuracy than larger filtered shallow networks.[1,2]

Therefore tiny group sizes and tiny filters need fast convent algorithms. However large batch sizes and large filtered are required for conventional convent libraries for fast operation.

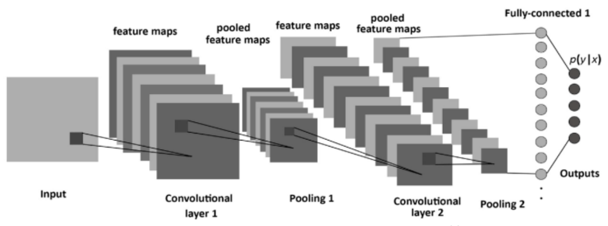
Minimal filtering algorithms pioneered by Winograd is the base on the convolution neural networks (convnets) [3] which is the new class of fast algorithm was introduced in this paper. The arithmetic complexity of a convent layer can be reduced to a factor of 4 by the algorithms than that of direct convolution. Generally, dense matrix multiplies of sufficient dimensions will perform the almost all the arithmetic efficiently, even though the size of the batch is very small. Compared to the conventional Faster Fourier Transform convolution algorithm the memory need will be low. By using at most 16mb of workspace memory, and for all batch sizes the throughput is measured by Sate of art , from 1 to 64, was achieved by NVIDIA Maxwell GPUs

## 2. Related Work

Convnet layers arithmetic complexity was reduced by the FFT and convolution theorem, first by Mathieu *et al*,[4] visalache *et al* [5] refined it and in NVIDIA cuDNN library[6] it was implemented. To reduce the convolutions in a convent layer cong and xiao [7] used the Strassen algorithm for fast matrix multiplication and arithmetic complexity was reduced. For reducing the complexity of convents by quantizing several approaches were attempted. These approaches can be considered as complementary and orthogonal to those that exploit algebraic structure, and therefore those are not included in the span of this paper.

## 3. CNN (Convolutional Neural Network)

Convolution neural network is a deep learning network architecture. It learns from images directly. The CNN is made of several layers that processing transform an output to produce from an input. In CNN is trained to image analysis tasks including object detection, segmentation, scene classification and image processing. In Fig A where we can see the different types of stack layers each layers can presides and transforms the input volume into the output volume through distinguishable function using of convolution layer.



**FigA:** CNN processing the input images in different channels

A bank of K filters with size P × R and C channels can be correlated with a minigroup of N no.of images with size H ×W and C channels using the convolution layer. Filter elements are denoted by $G_{k,c,u,v}$ and $D_{i,c,x,y}$ gives image elements.

Th formula for computation of an single convolution layer ouput $Y_{i,k,x,y}$ is given as:

$$Y_{i,k,x,y} = \sum_{c=1}^{C}\sum_{v=1}^{P}\sum_{u=1}^{R} D_{i,c,x+u,y+v} G_{k,c,u,v} \tag{1}$$

The entire image/filter pair output can be written as

$$Y_{i,k} = \sum_{c=1}^{C} D_{i,c} * G_{k,c} \tag{2}$$

where 2D correlation is denoted by $*$.

## 4. High computing algorithoms

last few decades the outputs were computed using minimal filtering(MF) algorithm by including a p-tap FIR filter, which we describe (S, P), claims duplications .

$$\mu(F\ (s,p)) = s + p - 1 \tag{3}$$

Further, 1D algorithms can nest least F (s, p) and F (e, r) to develop least 2D algorithms for computing s× p outputs with an p × r filter, which we describe as
F (s × p, e × r). This claim

$$\mu(F\ (s \times p,\ e \times r)) = \mu(F\ (s,\ p))\mu(F\ (e,\ r)) \tag{4}$$

$$=(s - 1)\ (e + r - 1)$$

Duplications. Multi-dimensional (MD) FIR filters can be developed by nest 1D algorithm. It is fascinating to see that in multi-dimensional, 2D and 1D the least algorithm requires number of inputs must be equal to the numbers of the multiplications, i.e., to compute F (s, p) we must obtain an period of s + p − 1 input values, and to compute F (s × p, e × r) we need a path tile of (s + p − 1) × (e + r− 1) values of data. Hence the algorithm of minimal filtering, per input it needs one repetition.

The usual algorithm for F(2,3) does 3×2 = 6 multiplications. Winograd [3, p. 43] documented the resulting least algorithm (minimal algorithm):

$$F(2,3) = \begin{bmatrix} x_0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} s_1 + s_2 + s_3 \\ s_2 - s_3 - s_4 \end{bmatrix} \tag{5}$$

Where

$$s_1 = (x_0 - x_2)g_0 \qquad s_2 = (x_1 + x_2)\frac{g_0+g_1}{2}$$

$$s_4 = (x_1 - x_3)g_2 \qquad s_3 = (x_2 - x_1)\frac{g_0 - g_1}{2}$$

This algorithm does simply four matrix multiplications. Hence least (minimal) by the equation μ(F(2,3)) = 2 + 3 − 1 = 4. Further does four additions including the data, two multiplications and three additions by a constant meaning filter (at once the amount g0 + g2 can be estimated), and the outputs can be diminished by four additions at the final result. Matrix form of Fast Filtering Algorithms can be formulated as:

$$Y = A^T[(Gg) \odot (B^T x)] \tag{6}$$

Multiplication (element-wise) was indicated by ⊙. For F (2, 3), the matrices are

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T$$

$$x = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \end{bmatrix}^T$$

A least 1D algorithm F(s, p) is itself nested among to achieve a least 2D algorithm (minimal), F(s× s, p × p) similar to:

$$Y = A^T[[G_g G^T] \odot [B^T x B]]A \tag{8}$$

Now g is an p × p filter and x is an (s + p − 1) × (s+p−1) image tile. This nesting method can be generalized for non-square filters and results, F(s× e, p × r), algorithm is nested for F(s, p) by an algorithm for F (e, r). F (3×3, 2×2) uses 16= (4×4) multiplications, whereas the conventional algorithm uses 36= (3×3×2×2).

Here is a computation complexity loss of 36/16 = 2.25. The data transform does thirty-two additions, the filter transform uses twenty-eight floating point guidance, and that inverse transform does twenty-four additions. Algorithms toward F(s×s, p×p) can be applied to estimate convent layers with s × s kernels. Every picture channel is split into tiles size of (s+p−1) × (s+p−1), with s− 1 portions overlay between neighbouring tiles, allowing
Q = ⌈H/s⌉⌈W/s⌉ tiles per carrier(channel), C. F(s×s, p×p) is then estimated for each filter and tile sequence in every carrier (channel), and outputs are calculated over all carriers (channels). Interchanging U = GgGT and V = BT xB, we have:

$$Y = A^T[U \odot V]A \tag{9}$$

Marking coordinates of tile as (~w,~x), we edit the convnet layer equation (2) during every particular image i,and filter k, and co-ordinate of tile (~x,~ y) as therefore we can decrease over C carriers (channels) that measured and calculated send into tranformed time, and simply then implement the inverse modify in the A channels to the sum. The cost of the inverse transform above the number of carriers (channels) is amortized.

$$Y_{i,k,\tilde{w},\tilde{x}} = \sum_{c=1}^{C} D_{i,c,\tilde{w},\tilde{y}} * G_{k,c}$$

$$Y_{i,k,\tilde{w},\tilde{x}} = A^T[U_{k,c} \odot V_{c,i,\tilde{w},\tilde{x}}]A \tag{10}$$

$$= A^T[\sum_{c=1}^{C} U_{k,c} \odot V_{c,i,\tilde{w},\tilde{x}}]A$$

$$M_{i,k,\tilde{w},\tilde{x}} = \sum_{c=1}^{c} U_{k,c} \odot V_{c,i\tilde{w}\tilde{x}} \tag{11}$$

We consider the sum and analyse the system by dropping the image/tile coordinates (i,~w,~x).into a one dimension, we also marked every component of the (element-wise) multiplication divider ,as (∑,v) Placed :

$$M_{k,b}^{(\varepsilon,v)} = \sum_{c=1}^{c} U_{k,c}^{(\varepsilon,v)} \odot V_{c,b}^{(\varepsilon,v)} \tag{12}$$

This formula is just matrix arithmetic, so that determined as a:

$$M^{(\xi,\nu)} = U^{(\xi,\nu)} V^{(\xi,\nu)} \tag{13}$$

Matrix multiply has accuracy development on Central Processing Unit (CPU), Graphical Processing unit (GPU), and Field Programmable Gate Arrays (FPGA)platforms, owing to its powerful computational intensity. Therefore we have reached at the effective implementation of the fast algorithm placed in below mentioned Algoritham 1 .
A method for producing the least filtering algorithm F(s,p) for every variety of s and p was documented by Winograd [3]. The chinese remainder theorem was used by the form to provide a least (minimal) algorithm for linear convolution, i.e. similar to polynomial arithmetic, then transfers the linear convolution algorithm to produce a least (minimal) filtering algorithm. We present sources of the specific algorithms applied in the additional material of this paper.

**Algorithm 1** Compute Convnet Layer with WMF (Winograd Minimal Filtering) Algorithm F (s × s, p × p)

```
    P = N [H/s][W/s] is the count of image tiles. α =( s + p – 1)
    is the size of input tile.

Surrounding tiles overlap by p − 1.
$d_{c,b} \in r^{\alpha\times\alpha}$ is input data image piping b in carrier (channel) c.
$g_{k,c} \in r^{p\times p}$ is filter that was applied k in carrier (channel) c.


G, $B^T$ , and $A^T$ are filter image input data, inverse transforms, and
data input for channels .
$Y_{k,b} \in r^{s\times s}$ is resulted image piping b in k filter.
for k = 0 to K do

for c = 0 to C do
            $u = Gg_{k,c}G^T \in R^{\alpha\times\alpha}$
            Matrices Spitted U: $U^{(\xi,v)}_{k,c} = u\xi,v$

        Fo9weightisr b = 0 to P do

        for c = 0 to C do
            $v = B^T d_{c,b} B \in R^{\alpha\times\alpha}$
            Spitted v to matrices V: $V_{c,b}^{(\xi,v)} = v\xi,v$

    for ξ = 0 to α do

        for ν = 0 to α do
            $M^{(\xi,v)} = U^{(\xi,v)} V^{(\xi,v)}$

    for k = 0 to K do
        for b = 0 to P do

            From matrices S s is gathered: $s_{\xi,v} = s_{k,b}^{(\xi,v)}$

            $Y_{k,b} = A^T S A$
```

## 4.1 Minimal 2D algorithm of F(3×3 with 2×2) :

Training a system utilizing highly inclination origin requires computation of the gradients including regard to the input data and (weights) loads. For a convolutional layer, the highest gradient by regard to the inputs data is a convolution of the resulting layer's back propagated fault, of dimension (H × W × N × K), including a flipped variant of the layer's S × R filters. Hence it can be estimated applying the corresponding algorithm that is employed for forwarding propagation.

The pitch with regard the loads (weights) is a layer data inputs convolution including the back propagated errors, providing S×R results carrier (channel) and per filter. Hence we require computing the convolution F (H×W, R×S), as W×H is important too huge for our active algorithms so which is impossible. Alternatively, we crumble this convolution into a linear sum of inadequate convolutions, for in a case, F (2 × 2, 3 × 3). Here the algorithm's 4 × 4 image tiles are overlain by 2 pixels in particular dimension, and the 3 × 3 results are summed over all tiles to form F( H × W,3 × 3).

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, G = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{bmatrix}$$
$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{14}$$

With $(3+2-1)^2 = (5-1)^2 = 16$ multiplies versus linear convolution's 2 × 2 × 3 × 3 = 36 multiplies, it gets the same 36/16 = 2.25 arithmetic difficult computing reduction as the same way forward propagation algorithm.

## 4.2 Minimal 2D algorithm of F(4×4 with 3×3):

The F (4, 3) is least (minimal) algorithm has the derived below
The data shifts do thirteen floating position directions; the filter shifts handle eight, and the inverse transform handle ten. Implementing the nesting equations allows the least (minimal) equation for F(4 × 4), F(3 × 3) that works thirty-six (36=6×6) multiplies, while the conventional algorithm uses one forty-four (4×4×3×3 = 144). This is a computational complexity loss of four. The two-dimensional data transform do's one fifty-six ((6 + 6)13 = 156) floating point directions, the filters are applied does seventy two ((3 + 6)8 = 72), and the inverse transform does one hundred (10(6 + 4) = 100). The amount of computing and constant multiplications needed by the least (minimal) Winograd converts increases quadratically with the tile size Therefore for

$$B^T = \begin{bmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{bmatrix}$$
$$G = \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{bmatrix}$$
$$A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{bmatrix} \tag{15}$$

The difficult to a processing in one form to another form will confuse any proceeds in the number of multiplications. The consequence of the transform matrix components also improves with progressing tile size.

### 4.4 Faster Fourier Transformation for CNN:

The Faster Fourier Transform can be applied to generate a piped convolution algorithm are the identical form as above mention Algorithm 1. The central variation is that the converts into the matrices form and restored among Fast Fourier Transform and inverse Fast Fourier Transform, and straight forward multiplication of complicated Fast Fourier Transform elements generates cyclic convolution. Only $s \times e$ elements of the $(s+p-1)\times(e+r-1)$ cyclic convolution are accurate, the residue must be abandoned, and the tiles must be projected by $p-1$ and $r-1$ in method to regenerate the rejected outputs. This method is assigned to as projection and protect mainly, for the discrete fourier transform of an $[\alpha \times \alpha]$ array of original values can be designed with an array of $\alpha\times(\lfloor\alpha/2\rfloor+1)$ complex group values.
For increasing complex numbers the conventional algorithm is applied, this function of equals $4(\lfloor\alpha/2\rfloor + 1)/\alpha > 2]$ uncommonly multiplies per every data input. Different method, i.e. to our understanding has not been used in convents, is to practice a fast algorithm to multiply complex numbers with three real multiplications [3]:

$$(a_0 + ia_1)(b_0 + ib_1) = [a_0b_0 - a_1b_1, i(a_0b_1 + a_1y_0)] \\ = [u_cv_a + u_av_c, i(u_av_c - u_bv_b)] \qquad (16)$$

Where

$$u_a = a_0 \\ u_b = a_0 + a_1,\ v_a = b_0 \qquad (17) \\ u_c = a_1 - a_0,\ v_b = b_1,\ v_c = b_0 + b_1$$

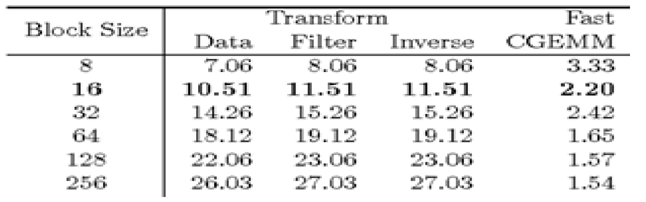
**Table 1:** Arithmetic complexity between Winograd algorithm Fast Fourier Transform

| Tile | | Winograd Algorithm | | | | FFT |
|---|---|---|---|---|---|---|
| | $a'$ | $b'$ | $c'$ | $d'$ | $a'$ | $d'$ |
| 3 | 8.00 | - | - | - | | |
| 4 | 5.00 | 3.00 | 4.75 | 1.50 | | |
| 5 | 1.78 | 2.60 | 3.24 | 1.24 | | |
| **6** | **2.25** | **4.33** | **2.00** | **2.78** | | |
| 8 | 4.78 | 6.50 | 1.23 | 3.38 | 3.44 | 4.42 |
| 16 | | | | | 4.94 | 3.23 |
| 32 | | | | | 6.42 | 4.24 |
| **64** | | | | | **2.20** | **8.30** |
| 128 | | | | | 5.14 | 13.37 |
| 256 | | | | | 1.05 | 11.42 |

In Table 1 multiply inputs tiles (a′), data transform forms the one tiles to another (b′), filter transform applied to data (c′), and inverse transform to normalized the data(d′) normalized arithmetic complexity between Winograd algorithm Fast Fourier Transform based convolution network of . Direct convolutions have tile size three.
A convnet algorithm based on Fast Fourier Transform can combine this by adjusting the FFT of the filter and data to result in the matrices with real values $(V_a,V_b,V_c)$ and $(U_a,U_b,U_c)$ instead of the obsession valued matrices V and U

$$T = U_aV_c\ M_1 = -U_bV_b + T, \\ M_0 = U_cV_a + T\ M = (M_0, iM_1) \qquad (18)$$

**Table 2:** The Stability and Normalization between the Fast Fourier Transform and CGEMM



| Block Size | Transform | | | Fast |
|---|---|---|---|---|
| | Data | Filter | Inverse | CGEMM |
| 8 | 7.06 | 8.06 | 8.06 | 3.33 |
| **16** | **10.51** | **11.51** | **11.51** | **2.20** |
| 32 | 14.26 | 15.26 | 15.26 | 2.42 |
| 64 | 18.12 | 19.12 | 19.12 | 1.65 |
| 128 | 22.06 | 23.06 | 23.06 | 1.57 |
| 256 | 26.03 | 27.03 | 27.03 | 1.54 |

The collection of brief matrix T is achieved using natural Single precision floating General Matrix Multiply(SGEMM) with C = T and $\beta = 1$, at that value of computing two floating period guidance per output. In the Table 2 Think of these preparations as joining to the cost of inverse transform. As we compares the Fast Fourier Transforms and Single precision floating General Matrix Multiply (SGEMM).

## 5. Logical Computing Analysis

In our system of fast convnets, the arithmetic stage computation complexity is:

$$B = N\lceil H/s\rceil\lceil E/e\rceil CK(s+p-1)(e+r-1) \qquad (19)$$

If s = e = 1, equation approaches the arithmetic complexity of linear convolution. Hence linear convolution is the least algorithm for $F(1 \times 1, P \times R)$, although our report applies least(minimal) convolutions, the convnet layer itself means claim not least(minimal) because it produces higher convolutions than are rigidly required. We could decrease the number of convolutions by applying Strassen recursions(SR Recursions) as in **[6]**, but every recursion decreases all three dimensions of our models by half while contributing only an (8/7) decrease in computation complexity.
In classification to easily the formulas, we will pretend that W/s and H/e have no remains. This multiplication complexity has derived as:

$$B = (s + p - 1)2/s2NHECK \\ = \alpha' NHECK \qquad (20)$$

Thus $\alpha = (s + p - 1)2$ and $\alpha' = \alpha/s2$ the total equations that can be translated and further equation

$$T(D) = \beta/s2NHEC,\ T(F) = \gamma CK,\ T(I) = \delta/s2NHEK \qquad (21)$$

Wherever $\delta$, $\beta$ and $\gamma$, is the amount of floating point directions applied by the relative transforms for individual tiles. Distributing the difficulty of every change by in M allows its corresponding complexity:

$$T(D)\ B/ = (\beta)/(K\alpha^2) = (\beta')/K \\ T(F)/B = \gamma/(NHE\alpha^2/s^2) \\ = \gamma/(P\alpha^2) = \gamma'/P \qquad (22) \\ T(I)/B = \delta/(C\alpha^2) = (\delta')/C$$

We define as $\gamma'$, $\delta'$, and $\beta'$ the normalized computation complexities of the filter, input data, and the corresponding inverse transforms, individually. $P = HNE/s2$ is the no. of pipeline per channel as in the unit by Average multiplication complexity of the content layer is given by combining the intervals for each stage:

$$Q = (1 + \beta'/K + \gamma'/P + \delta'/C)\ \alpha'\ (NHECK) \qquad (23)$$

In series to get a large boost up, the arithmetic complexity of $\alpha'$ is very small then the transform complexities $\delta'$ $\beta'$, and $\gamma'$ need each be short correlated with C, P and K individually.
Based convnet layers and FFT including fast CGEMM is highly effective with the Winograd algorithms is most faster then CGEMM, the tile size is higher compared to Winograd means Faster Fourier Transform based convnet design to hold modified data required a high representation workspace. To transform cost a common amount of transformed data must be held in position and to produce matrices with high accuracy of with CGEMM that GPU needs high computational for the convolutions layers to read and processing data.

## 6. Implementation of GPU

We implemented F(2 × 2,3 × 3) the newest formation of NVIDIA GPU[7], the 1080Ti(titan X), has many benefits over GPU for computational multiple prominently the introduction of a R/W L2 global cache for device memory. This allows accelerated programs and analyses signing the code. In case, the similar change of difficulty into device relieves various software and optimization difficulties. Our operations note that the CNN performance grows 2-3 generations faster simply by changing from GPU 1080ti to 1085ti Conventional evaluations of CUDA code is extremely time spending and incompetent inclined. We Computing processing for the innovative architecture, relying on the L2 cache for many of the device representation access, alternatively of input writing code that uses arrangements and shared memory.in below table 3 diffrentional layers of VGG network with a gflops standard optimizations are present future image analysis problems.
Code received by this Masic strategy is fast adequately. We use the back types of Computing: pre-computed representations presented loops within template kernels, stamped patterns to achieve combined vision accesses and personnel wheresoever possible.In Table 3 Supplementary standard optimizations are pleasant in case future image analysis problems will demand even more computing power that depends on computations of both GPU.

## 7. Experimenting on different networks

For More accuracy and speed experiments with VGG Network **[2],** the data VGG is tested in Google Tensorflow and are managed by us. The 3x3 filters with a deep network in the convolution layers, are tabulated in the Table 3.

**Table 3:** Convolution layers of VGG network of the point defined

| Layers | Depth's | C × W × H | | K | GFLOPs |
|---|---|---|---|---|---|
| conv 1.1 | 1 | 64 × 234 × 234 | | 64 | 0.18 |
| conv 1.2 | 1 | 63 × 234 × 234 | | 64 | 2.70 |
| conv 2.1 | 1 | 64 × 275 × 132 | | 128 | 2.85 |
| conv 2.2 | 1 | 128 × 132 | × 132 | 128 | 1.70 |
| conv 3.1 | 1 | 128 × 58 | × 58 | 256 | 2.85 |
| conv 3.2 | 2 | 256 × 58 | × 58 | 256 | 10.10 |
| conv 4.1 | 1 | 253 × 58 | × 58 | 512 | 2.85 |
| conv 4.2 | 2 | 512 × 38 | × 38 | 512 | 10.10 |
| conv 5 | 3 | 512 × 15 | × 15 | 512 | 13.70 |
| | | | | | |
| Total | | | | | 47.03 |

Our fast algorithms with single (fp32) and half (fp16) precisions data and filters are observed for accuracy .fp32 arithmetic instructions are used by us. The uniform distribution [-1, 1], consisting of random data and filters are used to calculate the absolute element error.By direct convolution with a double precision accumulator, we analyze ground truth for any changes.
Our GPU implementation of F(2x2, 3x3) in correlation with cuDNNv3**[1]** measures its speed on a superclocked NVIDIA Titan X GPU.A maximum clock rate of 1126 MHz was observed by disabling the boost clock. A GPU yields a device peak throughput of 2 × 3072 × 1126 = 6.96 TFLOPS and has 3072 cores..Division of computation of number of GFLOPs by direct convolution gives the speed for a given layer which is listed in table-3.
This Algorithm is active on the GPLOP. The ratio of the total GPLOPs and runtime Code gives total throughput.

## 8. Results and Discussion

The single precision (fp32), half precision (fp16) and their filters Are used to test the accuracy of the different convolution layer algorithms with input data .F(2 × 2, 3 × 3) is more accurate than that of direct convolution. The simple transforms of F (2 × 2, 3 × 3) do not lose much precision, and reduction over C channels is offered by its multiplication stages, rather than RSC filter elements which are overcome by convoluting directly . F (4 × 4, 3 × 3) has a more substantial erroneousness, even though it is more accurate than convoluting directly with fp16 data.
The algorithm testing are reasonably accurate with fp16 data. Here the precision of the inputs is limited in accuracy. Convoluting directly is accurate for training and having deduction with data of low precision [4, 5].so, it has limited accuracy in the precision of the inputs. We conclude that in F (3 × 3, 4 × 4) .Table 6 and Table 5 shows the total throughput of VGG Networklayer-E for cuDNN and our F (2×2, 3×3) is implemented in fp16 and fp32 data for different sizes of batch. For fp32 data, F(2 × 2, 3 × 3) is 1.48X at N = 64 and 2.26X as fast at N = 1. The N = 16 has throughput of 9.49TFLOPS. For fp16 data, F(2×2, 3×3) extends its gain overcuDNN, recording 10.28 TFLOPS for N = 64's throughput.The performance of N=8 is pretty good at 9.57 TFLOPS.

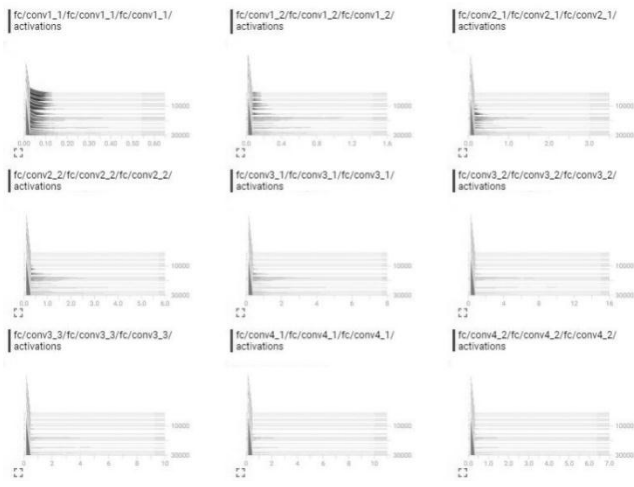


**Fig. 1:.:**Histogram of VGG fp32 data.

In Fig. 1 exposes layer throughput. If cuDNN uses the FFT algorithm, then hatch marks indicate the layer, otherwise direct convolution will be used. For F (2 × 2, 3 × 3), the external filter transform (FX) is used which are indicated by the hatch marks; otherwise, the fused transform will be faster.

cuDNN is likely to select its FFT algorithm for intermediate values of N erroneously, despite the fact that it performs poor, under 2 TFLOPS. It reveals that it's just a bug. Low performance at moderate values of N suggests that the convolution by FFT implementation either uses large tiles or just a single tile per image, asin [5]. Unless N is large this leads to inefficient multiplication stages.cuDNN FFT performs much better but stays well under 8 TFLOPS, at large N.F(2×2, 3×3) performs better than cuDNN at every layer and group size, except the layer conv1.1, which contributes less than 0.5% of the total network computation in general, we notice that the FX variant of our implementation performs best unless the number of filters and channels are substantial.
The filter transform computation requires large memory. Thus transformation of more massive filter decreases. The computational efficiency.F(2 × 2, 3 × 3) performs inadequately in the case when N=1 for 14×14 layers. During this case, the 8×4 superblockruns over the image boundary and computes unwanted pixels. The output of this layer configuration is about 5TFLOPS, where cuDNN performance is just 1.6 TFLOPS.
A global memory workspace up to 2.6GB is used by cuDNN FFT in our experiments. In contrast, our fused F((2×2)×( 3×3)) implementation global workspace not used, and only 16 MB was used by the FX variant.

**Table 4:** fp32 versus Fp16

| Layers | fp32 | | | Fp(16) |
|---|---|---|---|---|
| | Directs | F(2x2),F(3x3) | F(4x4),F(3x3) | |
| 1.2 | 2.01E-05 | 2.43E-05 | 2.84E-04 | 1.14- 02 |
| 2.2 | 5.11E-05 | 3.56E-05 | 4.41E-04 | 2.45E-02 |
| 3.2 | 2.43E-04 | 4.44E-05 | 8.06E-04 | 2.99E-02 |
| 4.2 | 4.10E-04 | 3.44E-05 | 1.05E-03 | 4.17E-02 |
| 5 | 2.13E-04 | 3.50E-05 | 1.07E-03 | 5.61E-02 |

In above Table 4 new capabilities for high output is shown by performance of F (2 × 2) × (3 × 3) and small group size with state of the art convolutional neural networks. When F (4 × 4, 3 × 3) is used, we expect the performance to improve. The maximum number errors that Occurs in VGG networks Layers

**Table 5:** VGG Versus Nvidia cuDNN In F32 data

| T layers | Nvidia cuDNN | | F(2x2,3x3) | | Speedup |
|---|---|---|---|---|---|
| | M(sec) | TFLOPS | msec | TFLOPS | |
| 1 | 13.22 | 2.12 | 6.45 | 1.23 | 1.96X |
| 2 | 30.46 | 4.83 | 8.89 | 6.49 | 2.26X |
| 4 | 106.40 | 2.49 | 18.72 | 8.81 | 4.51X |
| 8 | 331.01 | 0.89 | 45.11 | 9.43 | 6.24X |
| 16 | 210.01 | 4.03 | 12.79 | 8.49 | 2.01X |
| 32 | 231.01 | 1.45 | 140.36 | 8.47 | 0.97X |
| 64 | 380.95 | 5.44 | 244.48 | 8.99 | 1.78X |

In above table-5 VGG versus Nvidia cuDNN in performance network to define the System f32 data.this different between the two structured functions between the two data formats. For fp16 data, F(2×2, 3×3) extends its gain overcuDNN, recording 9.22 TFLOPS for N = 64's throughput.The performance of N=8 is pretty good at 13.22 TFLOPS.

**Table 6:** VGG Versus Nvidia cuDNN In F16 data

| t layers | Nvidia cuDNN | | F(2x2,3x3) | | Speedup |
|---|---|---|---|---|---|
| | M(sec) | TFLOPS | M(sec) | TFLOPS | |
| 1 | 12.22 | 1.22 | 5.55 | 0.13 | 0.96X |
| 2 | 32.16 | 5.43 | 9.19 | 5.29 | 1.36X |
| 4 | 206.45 | 5.39 | 15.22 | 2.81 | 3.41X |
| 8 | 241.41 | 1.49 | 15.71 | 5.43 | 5.74X |
| 16 | 230.21 | 3.02 | 14.29 | 6.19 | 1.21X |
| 32 | 241.21 | 2.35 | 160.46 | 8.17 | 1.47X |
| 64 | 340.45 | 5.34 | 224.48 | 9.29 | 1.78X |

In above Table 6 VGG versus Nvidia cuDNN in performance Network layer is defined the system of fp16.cuDNN FFT performs much better but stays well under 8 TFLOPS, at large N.F (2×2, 3×3) performs better than cuDNN at every layer and group size, except the layer conv1.1, which contributes less than 0.5% of the total network computation in general, we notice that the FX variant of our implementation Performs best unless the number of filters and channels are Substantial.

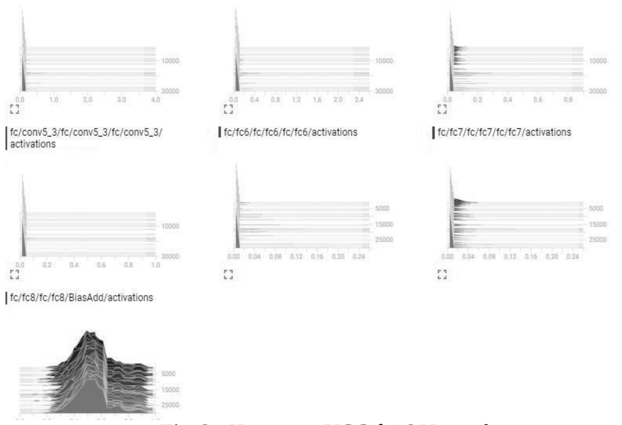


**Fig. 2:.** Histogram VGG fp16 Network

In above Fig. 2 the tensorflow data which will be prescribe faster relevant the time during training with different layers in processing above the mean time graphs will be regulated

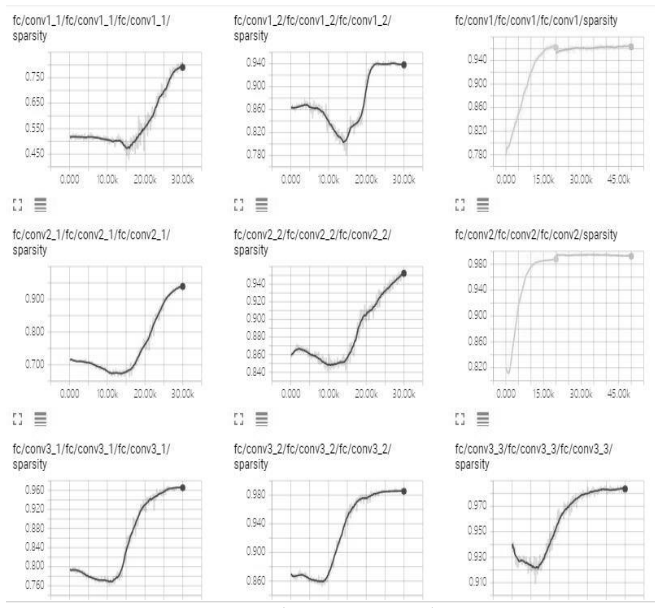


**Fig. 3:** The sparsity of two networks Of VGG network

In above Fig-3 cuDNN each graph that draw in the following with respective of the functional time period. This time period that represent the training part of each data which can allocate by the processing by differentiate functions and the equations. A global memory workspace up to 2.6GB is used by cuDNN FFT in our experiments. In contrast, our fused F ((2×2) × (3×3)) implementation global workspace not used, and Only 16 MB was used by the FX variant

## Reference:

[1] Christian Szegedy and Sergey Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", 2015 In *arXiv*: 1502.03167.

[2] R. K. Srivastava, K. Greff, and J. Schmidhuber. "Training very deep networks". 2015. *arXiv*:1507.06228.

[3] S.Winograd. "Arithmetic complexity of computations", volume 33. *Siam*, 1980:241-248

[4] ann LeCun." Fast training of convolutional networks through ffts". 2014, *Computer Research Repository*, abs: 1312.5353.

[5] Antonio J. Plaza, Chein-I Chang "High Performance Computing in Remote Sensing" CRC-2015 210-222.

[6] cuDNN http://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html acessed on 2018 March .

[7] Jason Cong and Bingjun Xiao. "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks." volume 24.*Computer Vision and Pattern Recognition*.2015 282-294.