

Understanding Gram Matrices

(Style Transfer)

Batchu Venkat Vishal

A gram matrix captures the overall style information of a given input by calculating a 2D matrix. In style transfer we use gram matrices to compare and regulate the style losses between our target style image and the generated image. A gram matrix is essentially a kernel matrix which helps represent style in a simple manner.

To compute a gram matrix of a given $C \times H \times W$ input volume (the output of some convolutional layer) where C = number of layers in the input volume, H = height of each layer and W = width of each layer. The gram matrix returns a $C \times C$ matrix which captures the style between all pairs of layers. First we flatten out $H \times W$ to get a 1D vector of length $H \times W$ and then our input volume is essentially a 2D matrix of dimensions $(C, H \times W)$. We then choose 2 layers at a time from the C layers, let's call them i, j and then we compute the dot product of the vectors at those layers. This would give us a single number which corresponds to the relation between those 2 layers. We then store this as,

$$\text{GramMat}[i][j] = \text{dot}(\text{transpose}(\text{Input}[i]), \text{Input}[j])$$

Where $\text{Input}[i]$ and $\text{Input}[j]$ are $H \times W$ sized vectors

We can observe that the Gram matrix is a symmetric matrix and it also satisfies all the other properties of Kernels.

In Torch, we can achieve this by creating a network which takes as input a (C, H, W) dimensional input volume and then flatten it out to create a $(C, H \times W)$ matrix and then make a copy of it to get two matrices $[(C, H \times W), (C, H \times W)]$ and then finally compute the dot product between them transposing the second matrix to get a (C, C) dimensional matrix which is our gram matrix. (Attached screenshots denoting the three steps below)

```

function GramMatrix()
    local net = nn.Sequential()
    -- Set input as batches with the last 2 dimensions belonging to each input.
    -- The last 2 dimensions are merged and flattened into a 1D array
    net:add(nn.View(-1):setNumInputDims(2))
    return net
end

img = torch.randn(5,3,3):add(3)
GMat = GramMatrix()
print (GMat:forward(img))

```

```

vishalapr@vishal-Lenovo-G50-70: ~
File Edit View Search Terminal Help

vishalapr@vishal-Lenovo-G50-70:~$ th test.lua
3.8643 3.3903 3.0635 3.9608 4.7832 3.5950 0.7586 1.9368 3.2322
2.6584 2.9724 2.0836 2.4229 3.6774 4.4963 2.5273 3.1307 3.6753
2.4049 3.5701 1.3723 1.2304 0.9662 4.3389 3.1029 2.8232 3.9627
2.3978 3.4149 2.8230 1.3928 2.9779 3.5171 1.6486 5.5078 3.3148
3.8273 5.3773 2.2030 3.9102 0.8653 4.1761 2.7115 2.1811 2.2568
[torch.DoubleTensor of size 5x9]

```

```

function GramMatrix()
    local net = nn.Sequential()
    -- Set input as batches with the last 2 dimensions belonging to each input.
    -- The last 2 dimensions are merged and flattened into a 1D array
    net:add(nn.View(-1):setNumInputDims(2))
    -- A concat table essentially takes the input given to it and runs the
    -- different modules in the concat table in parallel.
    local concat = nn.ConcatTable()
    -- Identity passes the input as the output without any changes
    concat:add(nn.Identity())
    concat:add(nn.Identity())
    net:add(concat)
    return net
end

img = torch.randn(5,3,3):add(3)
GMat = GramMatrix()
print (GMat:forward(img))

```

```

vishalapr@vishal-Lenovo-G50-70: ~
File Edit View Search Terminal Help

vishalapr@vishal-Lenovo-G50-70:~$ th test.lua
{
  1 : DoubleTensor - size: 5x9
  2 : DoubleTensor - size: 5x9
}

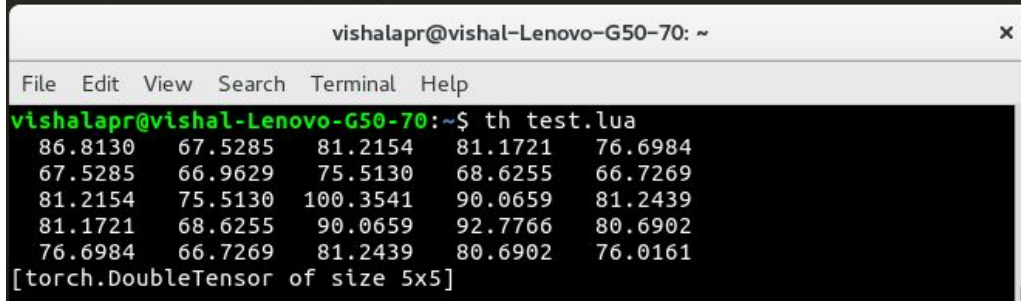
```

```

function GramMatrix()
    local net = nn.Sequential()
    -- Set input as batches with the last 2 dimensions belonging to each input.
    -- The last 2 dimensions are merged and flattened into a 1D array
    net:add(nn.View(-1):setNumInputDims(2))
    -- A concat table essentially takes the input given to it and runs the
    -- different modules in the concat table in parallel.
    local concat = nn.ConcatTable()
    -- Identity passes the input as the output without any changes
    concat:add(nn.Identity())
    concat:add(nn.Identity())
    net:add(concat)
    -- MM multiplies two input matrices by computing a dot product. The parameters
    -- false, true indicate if the matrices should be transposed or not.
    net:add(nn.MM(false, true))
    return net
end

img = torch.randn(5,3,3):add(3)
GMat = GramMatrix()
print (GMat:forward(img))

```



```

vishalapr@vishal-Lenovo-G50-70: ~
File Edit View Search Terminal Help
vishalapr@vishal-Lenovo-G50-70:~$ th test.lua
 86.8130   67.5285   81.2154   81.1721   76.6984
 67.5285   66.9629   75.5130   68.6255   66.7269
 81.2154   75.5130  100.3541   90.0659   81.2439
 81.1721   68.6255   90.0659   92.7766   80.6902
 76.6984   66.7269   81.2439   80.6902   76.0161
[torch.DoubleTensor of size 5x5]

```