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 x H x 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 x C matrix which captures the style between all pairs of layers. First we flatten out H x W to get a 1D vector of length HxW and then our input volume is essentially a 2D matrix of dimensions (C, HxW). 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,

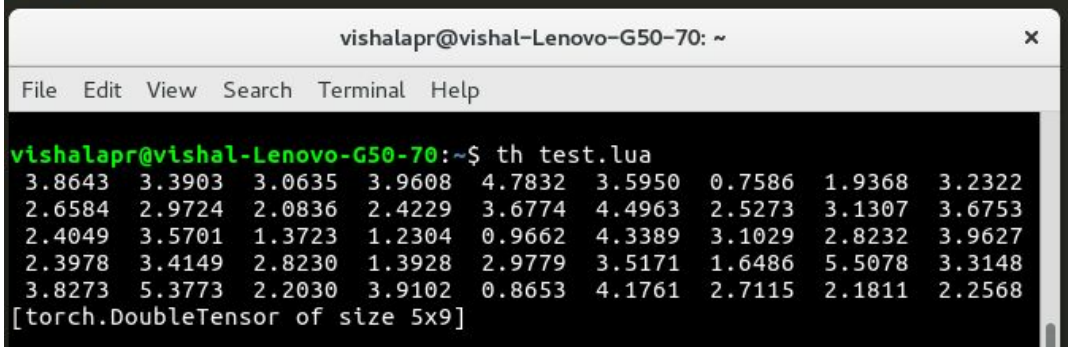
GramMat[i][j] = dot(transpose(Input[i]),Input[j])
Where Input[i] and Input[j] are HxW sized vectors

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,HxW) matrix and then make a copy of it to get two matrices [(C,HxW),(C,HxW)] 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)

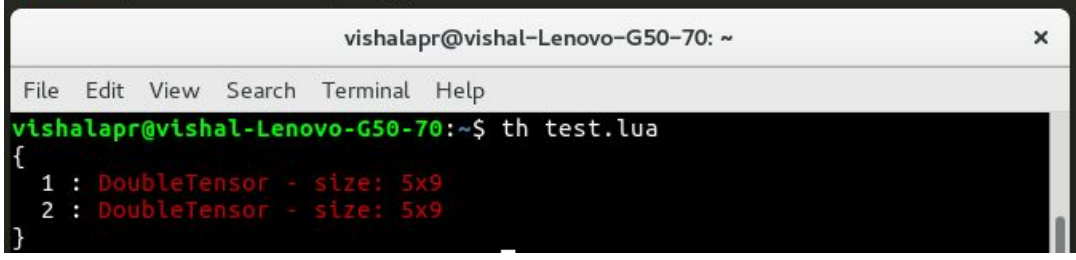
```lua
function GramMatrix()
  local net = nn.Sequential()
  -- Set input as batches with the last 2 dimensions belonging to each input.
  -- The last 2 dimensions are merged and flattened into a 1D array
  net:add(nn.View(-1):setNumInputDims(2))
  return net
end

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




```lua
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))
```



```lua
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))
```