Image Style Transfer

Problem Statement:
Combine style and content from different images to generate a new image.

The main idea behind the paper is that object recognition conv nets are often inherently good at separating style of an image from the content of an image. This is because object recognition mainly deals with the shapes and arrangement rather than the style of the object, so on being trained these models often help us separate “content” of an image from the “style” of an image. This is the main intuition behind why we use these conv nets to perform neural style.

The initial layers of the network capture the pixel level information but as we move on, the layers start developing representations of higher level information such as patterns etc. So it is important for us to evaluate and calculate the losses at various points in the network and not just at one of the layers since we want to capture the styles from the input style image at various levels in the network.

Overview:

1. Select a trained CNN model to use
    a. “**Imagenet-VGG**” initially but will also try with “GoogleNet” and “AlexNet” later on to compare and evaluate.
2. Select a framework to use for working with the network
    a. “**Torch** - LuaJIT”
3. Select a few layers from the network after which we add style and content loss modules
    a. 3-5 layers for **style loss** evaluation
    b. 1-2 layers for **content loss** evaluation
    c. (We would be modifying these to compare and evaluate the results as well later on)
4. We would write our own style and content loss modules which we would add after the selected layers in the network.
    a. We will define these modules in more detail below

Training noise to generate image:

Starting with a white noise image (or the content image itself with some noise) and then forward the image through the network so each layer would have the outputs and inputs stored and computed. Let the output image be ‘y’ and the random input image we use be ‘x’.

We then set the output gradient, dy = 0 and then start back propagating. So near the output layer of the network the gradients would be set to 0 but as we keep moving backwards slowly gradients would start accumulating due to the custom loss functions we define at the style and content loss layers. These gradients are then passed on backwards until we reach the input image where we modify it as x’ = x - dx*rate1.

We perform the above multiple times so that the overall error would keep getting reduced and after a few iterations (100-1000), we should have a decent image generated at the input which would have the style of the ‘style image’ (s_im) and the content of the ‘content image’ (c_im).

Content Loss Module:

1. We compute the features (c_f) of c_im (content image) at the input of the module and then store this to use as the target value. This is done by forwarding c_im through the network and taking the output at this layer.
2. We then compute the features (x_f) of our input x, at this layer by forwarding x through the network similar to what we did above. Now we compute the loss at this point using c_f and x_f. This would be a standard squared error loss.
3. Once we have this loss we sum it up along with appropriate weighting with the loss from the layer in front and then send this backward as the gradient at this module.

Style Loss Module:

1. Compute the gram matrix of features (s_f) of s_im (style image) at the input of the module and then store this to use as the style target value. We forward s_im through the network upto this point and then store this.
2. The gram matrix is a dot product of pairs of rows selected from s_f. This is used as a measure of the style of the image. This is what the researchers found as a good approximation to measure the style of an image. It makes sense since we consider different filters and compute their products along their depth (ie each value of that filter). So the style is the similarity between the various filters which we represent by the gram matrix.
3. Again similar to above, we add this loss along with appropriate weighting with the loss from the layer in front and then send this backward as the gradient at this module.

Note: The style loss module and the content loss module would have their targets computed initially as we specify the style and content images we would like to use to generate the output image starting with white noise as input.

Past/Current Research:

- Style Transfer has been extended to videos as well. Some papers directly apply this on videos frame by frame whereas others calculate differences between frames and apply it explicitly so it's a lot faster and doesn’t feel too noisy.
- A few methods similar to style transfer are being compared. http://cs231n.stanford.edu/reports2016/208_Report.pdf

- Style transfer on face portraits was done in 2014 itself, https://people.csail.mit.edu/yichangshih/portrait_web/ and the results were quite impressive. This was extended to videos as well.
- Generate almost painting like images (Crayon, wax, paint, pastels etc) - http://jaisrael.github.io/ParallelStyleTransfer/

Goals for evaluation 2: Complete paper implementation. Add TV loss functions. Apply various optimizations and experiment using different networks. Also try using sketches for the content and artworks for the style to try come up with an automatic sketch colouring.

Goals for final evaluation: Instead of using gram matrices, try coming up with something better. Improve on loss functions and try various changes to check if we can perform better than the current state of the art.

Work division:

- Vishal Batchu
  - Implement the loss modules
  - Generate modified network adding in these modules
  - Train initially using lbfgs, for basic outputs
- Raghuram Vadapalli
  - Various optimizations such as adam etc, to check which performs better.
  - Image pre and post processing to improve results
  - Try using different trained models for accuracy.
- Ameya Prabhu
  - Develop and formalize the theory which needs to be implemented.
  - Define functions mathematically which would be implemented.
  - Prove and figure out optimization methods which are worth trying and why they might work.
  - Come up with the theory and changes required to port this to sketch coloring.
  - Time based optimizations for the code to make it faster.