

CNN IMPLEMENTATION OF ARTISTIC STYLE

**GNR 652: MACHINE LEARNING FOR REMOTE SENSING
COURSE PROJECT**

OBJECTIVE

- To extract content and texture information from the images using a Deep Neural Network Model.
- To create a new image by using different proportions of the two features from two different images.

MOTIVATION

- To understand the natural phenomena of art sense and visual perception in humans
- To understand how the natural neurons interpret the texture of a visual observation and distinguish it from the content component of the visual source
- Texture relates to spacial similarity between pixels of an image, this project attempts to understand the behaviour similarity between natural neurons

IMPLEMENTATION

- VGG16 Convolutional Network is used for initializing the weights of the convolution layers
- The content and style losses are calculated at each layer and the total loss is taken as the weighted sum of two.
- The relative is decided as per requirement of extent of mixing.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 .$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

- A random image is initialized, say target image
- Input content Image and target image are passed through the content network
- The squared error is calculated for each layer and the weighted sum is denoted as the total content loss
- Similarly, for the input texture image with style network, total style error is the weighted sum.
- Total error is weighted sum of two errors, the weights are the user defined hyper-parameters

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

- Total error is minimized with variable values being the target image pixel values
- The minimization done by calculating the loss gradient wrt the pixels of target image using error back propagation

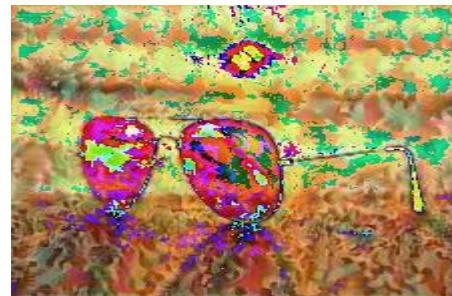
$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases} \quad \frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((F^l)^T (G^l - A^l))_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

OBSERVATIONS

- Observation made for $\alpha = 0.000008$ and $\beta = 0.00003$ with 200 epocs



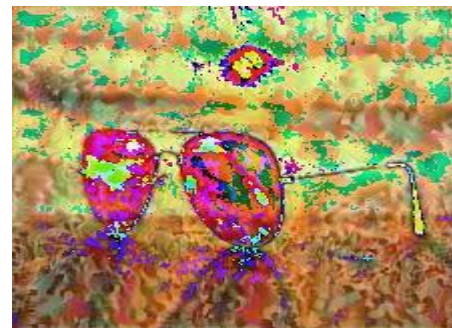
shutterstock.com • 1057977431



- Another made with same set of images, $\alpha = 0.00001$ and $\beta = 0.00002$ with 100 epocs



shutterstock.com • 1057977431



- The following set is made to understand the effect of increasing the number of epochs with $\alpha = 0.000001$ and $\beta = 0.00005$
- For number of epochs = 300



==

shutterstock.com • 1057977431



- For number of epocs = 1000



=

shutterstock.com • 1057977431



- For number of epocs = 2000



shutterstock.com • 1057977431



CONCLUSION

- The target image gets more similarity to the image input which has more weight to reduce the loss
- As the number of epochs increase, the target image gets more and more closer to the higher weighted input image
- Texture and content of the images cannot be totally isolated, therefore, the target image do show the features of both images

REFERENCES

- <https://www.robots.ox.ac.uk/~vgg/rg/papers/1508.06576v2.pdf>
- https://www.tensorflow.org/api_docs/python/tf