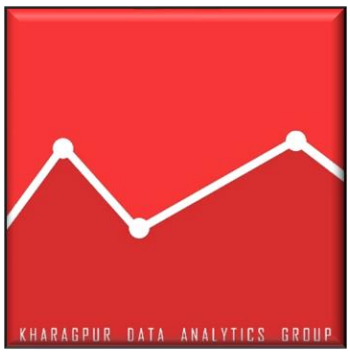


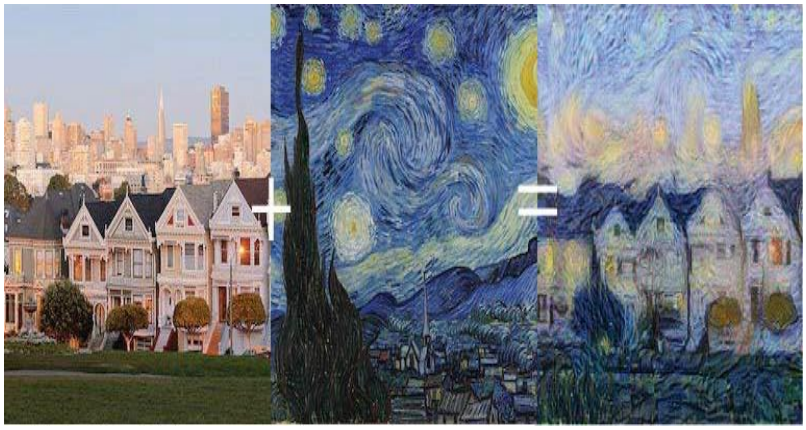
Task 2: Documentation On

Neural Style Transfer

Name: ***Amit Kumar Pal***

Roll No: **20EC10008**

What is Neural Style Transfer:



Neural Style Transfer is an interesting application based upon deep learning algorithms and image processing techniques, which results in the generation of a hybrid image which is superposition of two images, one of which is Content Image and other is Style Image. Depending upon the weights of these two images different combination of generated image can be produced.

NST (Neural Style Transfer) employs a pretrained Convolution Neural Network (CNN) to transfer styles from a given image to another. This is done by defining a loss function that tries to minimise the differences between a content image, a style image and a generated image.

Sample Output:

Content Image:

Style to be Inherited:

The style transferred image of above two is:

Content and Style Parameter for the above sample are:

Content Weight: 100: Style Weight: 1e8

Loss Functions associated with NST:

Content Loss:

It makes sure the content we want in the generated image is captured efficiently. **CNN** captures information about the content in the higher levels of the network, whereas the lower levels are more focused on the individual pixel values.

```
#content loss
def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))
```

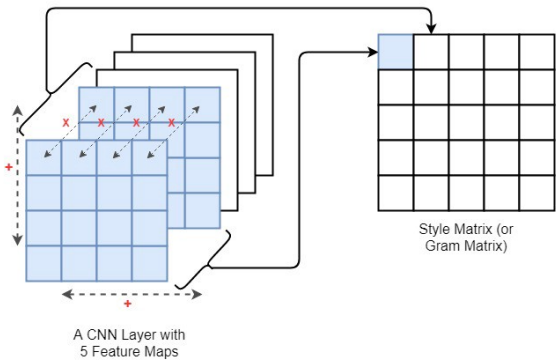
Here the base is the content features while the combination is the generated output image features. Here the **reduce_sum** computes the sum of elements across the dimensions of the specified parameters which is in this case the difference of corresponding pixels between input(content) and generated image.

Style Loss:

Style loss uses the squared loss function to measure the difference in style between the synthesized image and the style image. To express the style output of any style layer, we first use the extract_features function to compute the style layer output. Output into matrix X with c rows and h*w columns. This matrix can be thought of as the concatenation of c vectors x1, xc, each of which has a length of h*w. Here, vector xi represents the style feature of channel i.

Gram Matrix and its need:

In Mathematical terms, Gram Matrix is simply product of matrix X and its transpose X^T. In physical terms it generates correlation between style/colour/distribution extracted from style image to generated image. style is nothing but a correlation between shapes colour distributions etc so we can trick the network to extract style from feature matrices and the trick is known as the gram matrix. Gram matrix is a style representation that computes the correlation between different feature matrices which helps to capture the style of an image. Convolutional neural network on an image which will generate matrices which are called feature maps. Basically, these matrices contain low level feature of an image like lines edges dots curves and if we want to perform deep convolution, we will get some higher-level features like shapes objects etc. We know that different feature maps represent different feature of an image but these features are independent from each other so it can capture objects and shapes but cannot capture style of an image style refers to texture brush strokes colour distribution etc whose occurrences are dependent on each other. If we can measure the degree of correlation between these matrices, we can represent style of an image for instance when feature 1 occurs feature 3 occurs too so the correlation between feature maps can be calculated using dot products with respect to each other so in order to do that first convert the feature maps into vector and multiply it with its transpose hence the resulting matrix will represent style of an image this matrix is called the gram matrix.



Gram matrix contains style content, such as texture, colour and so on. Its definition is as follows:

$$Gij^{l} = \sum F_{ik}F_{jk}$$

Given the style picture $\vec{a}$, the target generated picture is $\vec{x}$, and the Gram matrix corresponding to the l layer is Al and Gl, respectively. The loss function of this layer is defined as follows:

$$El = (1/4N_l^2M_l^2) \sum (G_{ij}^{l} - A_{ij}^{l})^2$$

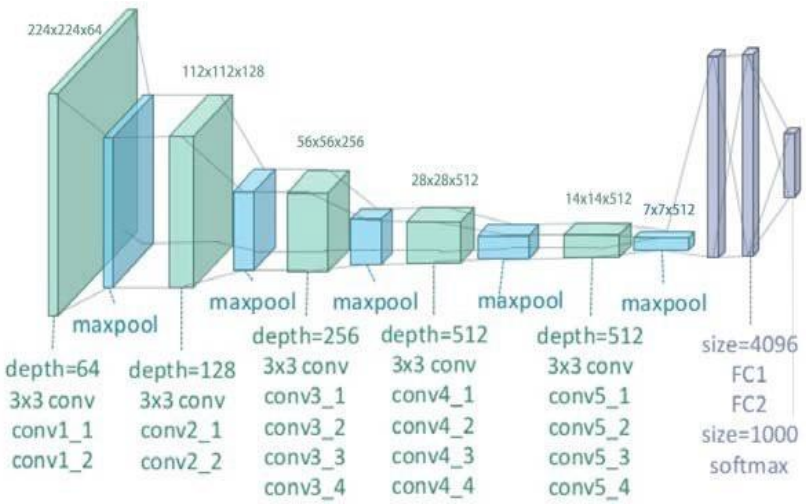
```
def gram_matrix(input_tensor):
    result = tf.linalg.einsum("fijc,fijd->fcd", input_tensor, input_tensor)
    input_shape = tf.shape(input_tensor)
    num_locations = tf.cast(input_shape[1] * input_shape[2], tf.float32)
    return result/(num_locations)
```

Excerpts from Papers:

- CNN consists of layers of small computational units that process information in feed forward manner. Higher layers learn about shape and objects, while lower layer learns about colour and visuals.
- The output of a given layer consists of so-called feature maps: Differently filtered version of input images.
- The input image is transformed into the representations that increasingly care about the actual content of the image compared to the detailed pixel values. Therefore, we are more interested into the feature response in the higher values.
- To obtain the representation of an input image, we use correlations between the different filter responses over the spatial extent of the feature maps. We obtain the multi-scale representation of the input image, which captures its texture information but not the global arrangement.

Algorithm of Artistic Style:

The content image will get into the network and will capture the feature from a little deeper convolutional layer of the network. This will give the complex feature of content image. In the same way we'll push style image into the network but at this time we'll capture feature from the first convolutional layer from every block first layer because it contains low level features of an image which we will need to get the style of an image so we'll store the features from every layer from for every block. Now the final step is to generate a white noise image and feed it into the network. Capture the features for content and style feature from their respective layers and in this way, we will have the generated output and the original output for both content and style.




```
In [26]: def style_content_loss(outputs):
             style_outputs = outputs["style"]
             content_outputs = outputs["content"]
             style_loss = tf.add_n([tf.reduce_mean((style_outputs[name] - style_targets[name])**2)
                                    for name in style_outputs.keys()])
             style_loss *= style_weight/ num_style_layers
             content_loss = tf.add_n([tf.reduce_mean((content_outputs[name] - content_targets[name])**2)
                                      for name in content_outputs.keys()])
             content_loss *= content_weight / num_content_layers
             loss = style_loss + content_loss
             return loss
```

Now we need to loss function, one which measures content loss and one which measures style loss so that we can perform back

propagation on image to reduce the loss as much as possible. So, the final loss function which we have to minimize is the summation of content and style loss.

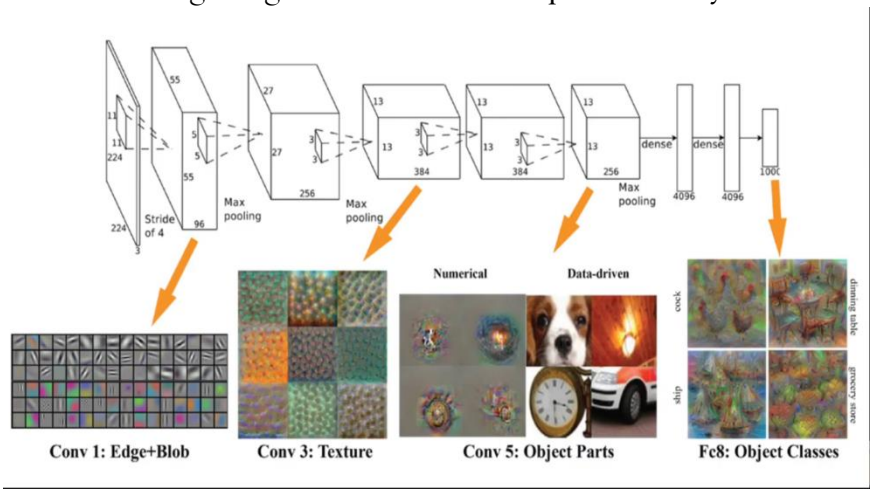
Staring algorithm follows the simple process that is simply reading the image, rescaling it to one dimensional array.

Using CNN VGG-19 in Neural Style Transfer:

VGG-19 is **a convolutional neural network that is 19 layers deep**. Using this we can load a pretrained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

After getting the pre process function for vgg-19, include top equals false represents that we are only downloading feature extraction part not the classifier. In the layer of the model there are no dense layer at the bottom neither the input layer on top now gets the name of every layer so that we can take the output from desired layer. Now here for every block there are multiple convolutional layers but we want the feature output for style is from first convolutional layer of every block and for content we can have feature output from second layer. When done, store the number of layers for content and style. Next, define a function to get the output from that layer only. Get the output layer from the model by using name of the layers finally attach those layers to models input and return it now use the above function to extract and store feature for style layers and feed the image to it and we can verify the output by checking shape from every layer. So, this part is the dimension of feature matrix and this is the number of feature metrics similarly we can get statistics for each layer and input for both style and for content.

After calculating the gram matrix for that particular layer now use the model to get the output.



Attach the output layer to the model and set style layer and content layer instance.

now write the flow for model. Store the output from each layer in the dictionary with the keys as layer name for both content and style layers.

Because the noise image will take more time to converge and the content image is already close to the content feature target. So, the model has to focus on minimizing the style loss more which will lead to faster convergence. Choose the optimizer to optimize the error and define the weight for both style and content for best balance in final image.

Now define the loss function. First get the output for both content and style and for style loss will implement mean squared error between output from noise and target output for output from every layer. Finally add both the style and content loss to complete the final loss equation. Train function is used to iteratively minimize the error.

First get the output for noise image and then measure the loss using above made loss function. Now use the gradient tape to measure the gradients and then use the optimizer to back propagate error to reduce the loss and normalize the image for next iteration.

Clear the current visualization from notebook and display the newly generated image from the tensor. We can see how the style is blending in this image.

Summarising the NST:

- The loss function commonly used in style transfer consists of three parts:

(i) content loss makes the synthesized image and the content image close in content features.

(ii) style loss makes the synthesized image and style image close in style features; and (

(iii) total variation loss helps to reduce the noise in the synthesized image.

- convolutional neural networks develop some complex understanding in form of feature matrices.
- Getting the style from those feature matrices directly is not possible that's why gram matrix is used to capture the styles.
- This method of style transfer is faster than any other training method because we are not training the model to do the style transfer

References:

- *https://iopscience.iop.org/article/10.1088/1757-899X/569/5/052107/pdf*
- *https://www.mathworks.com/help/deeplearning/ref/vgg19.html#:~:text=VGG%2D19%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals*.
- *https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee*
- *https://www.youtube.com/watch?v=Elxnzxk-AUk&t=4s*
- *https://www.youtube.com/watch?v=B22nIUhXo4E&t=2s*