INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD
Allahabad, Uttar Pradesh, India- 211012

Semester VII

Deep Learning Project Report

Theme based Colorization

In Partial Fulfillment of the Requirement for Degree of

Bachelor of Technology

In

Information Technology (B.Tech IT)

Under the supervision of: Dr Satish Kumar Singh

Submitted by :

Nehal Singh (IIT2018119)

Puja Kumari (IIT2018191)

Prabha Kumari (IIT2018195)

Akshat Solanki (IEC2018055)

TABLE OF CONTENT

ABSTRACT

We consider image transformation problems, where an input image is transformed into an output image which is a combination of the content image and style image. Recent methods for such problems typically train feed-forward convolutional neural networks using a per-pixel loss between the output and ground-truth images. Parallel work has shown that high-quality images can be generated by defining and optimizing perceptual loss functions based on high-level features extracted from pretrained networks. So In this project we are going to use convolutional neural networks for image transformation tasks and train using perceptual loss function with VGG19 model.

INTRODUCTION

Problem statement :

Implement an optimization technique using two images ,a content image and a style reference image (such as an artwork by a famous painter), and blend them together so the output image looks like the content image, but “painted” in the style of the style reference image.

Solution proposal:

In other key areas of visual perception such as object and face recognition, near-human performance was recently demonstrated by a class of biologically inspired vision models called Deep Neural Networks. Here we will implement an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality.
The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images.

MOTIVATION

In fine art, especially painting, humans have mastered the skill to create unique visual experiences through composing a complex image.There exists no artificial system with similar capabilities.

So, we are going to make such a model which can generate new images of high perceptual quality that combine the content of an arbitrary photograph with the appearance of numerous well known artworks.

content image + **style image** = **generated image**

Ancient city of Persepolis

The Starry Night (Van Gogh)

Persepolis
in Van Gogh style

LITERATURE SURVEY

We referred to a number of reports and papers to understand the extent of the problem areas.

1) **Paper title** : Perceptual Losses for Real-Time Style Transfer and Super-Resolution

   Name of Conference :

   Objective : Objective of this paper is to transform the input image into an output image which is a combination of the content image and style image.

   Methodology: Approach for solving image transformation tasks is to train a feed-forward convolutional neural network in a supervised manner, using a per-pixel loss function to measure the difference between output and ground-truth images. Such approaches are efficient at test-time, requiring only a forward pass through the trained network.

Result : They achieve comparable performance and drastically improved speed compared to existing methods, and to single image super-resolution where training with a perceptual loss allows the model to better reconstruct fine details and edges.

2) **Paper title** : A Neural Algorithm of Artistic Style

Name of Conference :

Objective : The key finding of this paper is that the representations of content and style in the Convolutional Neural Network are separable. That is, we can manipulate both representations independently to produce new, perceptually meaningful images. The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images.

Methodology: In this paper they introduce an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality. The results presented in the main text were generated on the basis of the VGG-Network, a Convolutional Neural Network that rivals human performance on a common visual object recognition benchmark task and was introduced and extensively described in. We used the feature space provided by the 16 convolutional and 5 pooling layers of the 19 layer VGGNetwork.

Result : They generate images that mix the content and style representation from two different source images.

SUMMARY OF PROPOSED PLAN

Our main aim will be to produce a mixed image out of two images provided, i.e a content image and a style image. Mixed image contains style (such as texture, colour) from style image and content from content image.

- Style Transfer : Technique of re-composing images in the style of other images
- CNN : The class of Deep Neural Networks that are most powerful in image processing tasks.
- We would like to present it with web-UI to make it user friendly.

PROJECT PLANNING

---

1. **Timeline**
    a. C1:

Requirement Verify	Done	5th September 2021
Project Planning	Done	8th September 2021
System Design	Done	20th September 2021
Details Design	Done	30th September 2021

b. C2:

Coding	Done	15th October 2021
Debugging and coding	Done	30th October 2021
Testing	Done	10th November 2021

c. C3:

Documentation and Final	Done	20th November 2021

2. **Dataset Description**

Our dataset consists of content image and style images .

For content images we will be taking different images from net

For style images we will take famous paintings like udnie , rain_princess etc.
We will be using pretrained VGG model :
http://www.vlfeat.org/matconvnet/models/beta16/imagenet-vgg-verydeep-19.mat

3. Technologies Used

a. python (Anaconda) [V 3.5.4]
b. tensorflow [V 1.4.0]
c. numpy [V 1.13.3]
d. PIL [V 4.2.1]
e. matplotlib [V 2.0.2]

4. Description of code availability

There are research papers and codes available which resemble our problem statement.
However most of them need a high GPU to run and cannot be run on our system. So we will be trying to implement the Gatys' Research paper.
There are some research papers which we use in order to understand algorithms.
Research paper :
[1508.06576] A Neural Algorithm of Artistic Style
Perceptual Losses for Real-Time Style Transfer and Super-Resolution
[1607.08022] Instance Normalization: The Missing Ingredient for Fast Stylization

5. Contribution of group

We took help from a research paper named "A Neural Algorithm of Artistic Style" (A Neural Algorithm of Artistic Style).

We are going to show this project through a web application for ease of use. There users can select content image image and style image and can see their output.

There are following tasks which did in this project :

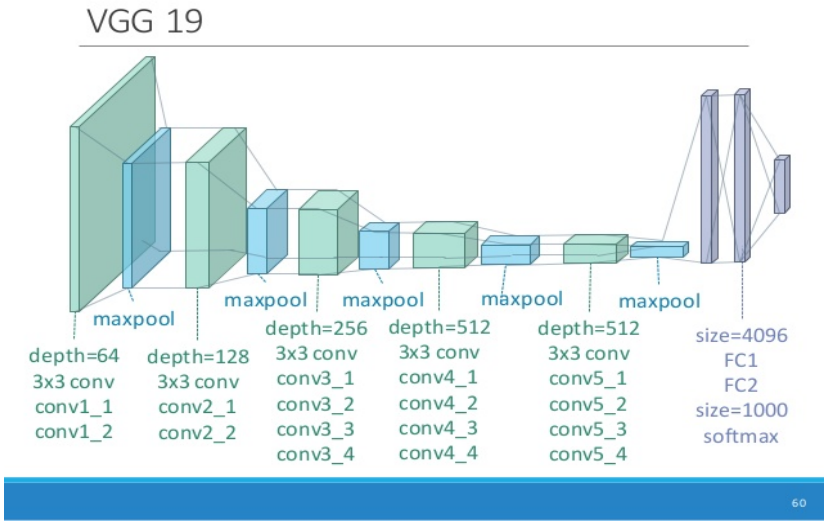
a. Training for Style Image
b. Testing for Style Image
c. Evaluation of checkpoint created using content image
d. Creation of web application

METHODOLOGY

CNN : CNN consists of layers of small computational units that process visual information hierarchy in a feed-forward manner

Each layer of units can be understood as a collection of image filters, each of which extracts a certain feature from the image.

Output of each given layer consists of feature-maps : differently filtered versions of the input image.



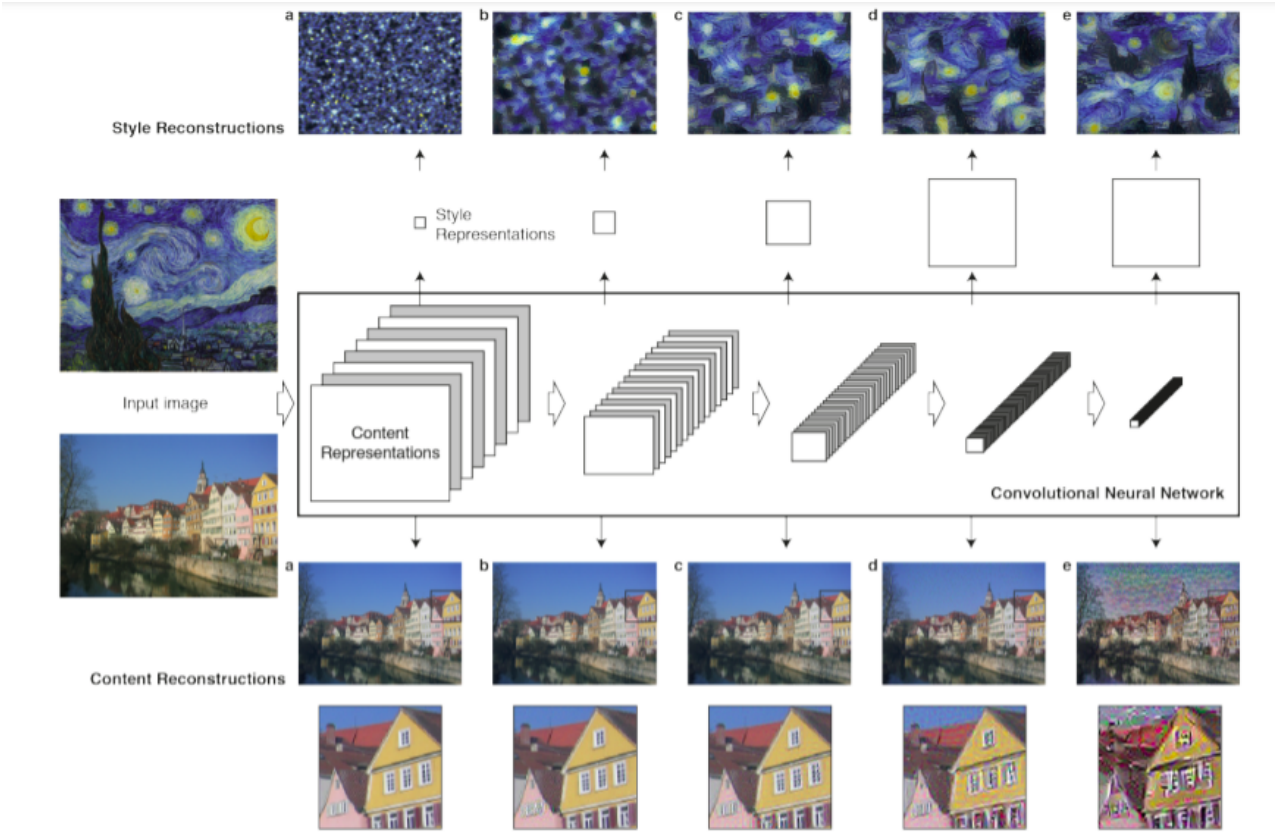
We use the intermediate layers of the model to get the content and style representations of the image. Starting from the network's input layer, the first few layer activations represent low-level features like edges and textures. As we step through the network, the final few layers represent higher-level features—object parts like wheels or eyes. In this case, we are using the VGG19 network architecture, a pretrained image classification network. These intermediate layers are necessary to define the representation of content and style from the images. For an input image we will try to match the corresponding style and content target representations at these intermediate layers. At a high level, in order for a network to perform image classification (which this network has been trained to do), it must understand the image. This requires taking the raw image as input pixels and building an internal representation that converts the raw image pixels into a complex understanding of the features present within the image.

This is also a reason why convolutional neural networks are able to generalize well: they're able to capture the invariances and defining features within classes (e.g. cats vs. dogs) that are agnostic to background noise and other nuisances. Thus, somewhere between where the raw image is fed into the model and the output classification label, the model serves as a complex feature extractor. By accessing intermediate layers of the model, you're able to describe the content and style of input images.

1. **Extraction of content from the image**
   The input image is transformed into representations that increasingly care about the actual content of the image as compared to its detailed pixel values.

   We refer to Feature responses in higher layers of the network as content representations.
   The ,4_2,5_2 convolution layer of the pre-trained VGG-19 network is used as a content extractor.

2. **Extraction of style from the image**
   "To obtain a representation of the style of an input image, we use correlations between the different filter responses over the spatial extent of the feature maps. We obtain a stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement"
   These correlations between feature maps are known as gram matrices.
   The layers used for calculation of the gram matrix are 1_1, 2-1, 3_1, 4_1, 5_1 with varied style weight constant for each layer. This constant can be seen as a hyperparameter used for changing style levels



3. The feature space provided by the 16 convolutional and 5 pooling layers of the 19 layer VGG Network.
4. $\vec{x}$ as input image , is encoded in each layer of the CNN by the filter responses to that image.
5. A layer with N_L distinct filters has N_L feature maps each of size M_L, where M_L is the height times the width of the feature map.
6. So the responses in a layer L can be stored in a matrix : $F^l \in \mathcal{R}^{N_l \times M_l}$

   where F^l_{ij} is the activation of the i^{th} filter at position j in layer L

7. **Content Loss**

   Let P be the original image and X , the image that is generated and P and F their respective feature representation in layer. We then define the squared-error loss between the two feature representations as the content loss.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F^l_{ij} - P^l_{ij}\right)^2$$

8. **Style Loss :** The style of an image can be described by the means and correlations across the different feature maps.
   Let A be the original image and X, the image that is generated and A_i and G_i their respective style representations in layer i.The contribution of that layer to the total loss is then

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

9. Loss :

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

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$ is Gram matrix ,where G_{ij}^l is the inner product between the vectorised feature map i and j in layer l .

10. To generate the images that mix the content of a photograph with the style of a painting , we jointly minimise the distance of a white noise image from the content representation of the photograph in one layer of the network and the style representation of the painting in a number of layers of the CNN.

IMPLEMENTATION

---

1. First step was to read content and style images.
2. If multiple style images are used , defining weight for each style image, else by default equal weights are assigned.
3. Defining:
   ```
   CONTENT_LAYERS = ('relu4_2', 'relu5_2')
   STYLE_LAYERS = ('relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 'relu5_1')
   ```
4. Neural_style.py is called with parameters "content image","style image", "checkpoints-output format","checkpoint iterations number"
5. The stylize() function from Stylize.py is called which yields tuples(iteration,image,loss_vals) at every iteration.
6. Normalization of style layer weights is done
7. Compute content features and Style features in feed forward mode
8. Make stylized image using backpropagation
9. Calculate content and style loss.

```
    content_losses.append(content_layers_weights[content_layer]*
    content_weight*(2*tf.nn.l2_loss(net[content_layer] -
content_features[content_layer]) /content_features[content_layer].size))
```

```
        content_loss += reduce(tf.add, content_losses)

style_losses.append(style_layers_weights[style_layer] * 2 *
tf.nn.l2_loss(gram - style_gram) / style_gram.size)
style_loss+=style_weight*style_blend_weights[i]*reduce(tf.add,style_loss
es)
```

10. Calculate total variation denoising and then total loss. Total variation denoising is decreasing high frequency artifacts by using an explicit regularization term on the high frequency components of the image.

```
tv_y_size = _tensor_size(image[:,1:,:,:])
tv_x_size = _tensor_size(image[:,:,1:,:])
tv_loss = tv_weight*2*((tf.nn.l2_loss(image[:,1:,:,:] -
image[:,:shape[1]-1,:,:])/tv_y_size)+(tf.nn.l2_loss(image[:,:,1:,:] -
image[:,:,:shape[2]-1,:])/ tv_x_size))
total_loss = content_loss + style_loss + tv_loss
```

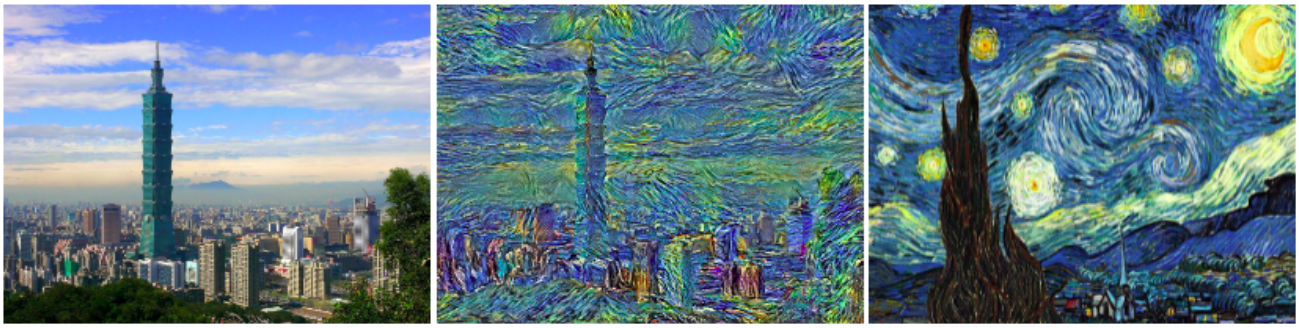
11. Setup optimizer as adam's optimizer and start optimization for each iteration. To optimize it we use a weighted combination of the two losses to get the total loss.

```
        train_step = tf.train.AdamOptimizer(learning_rate, beta1, beta2,
epsilon).minimize(loss)
```

12. Luminosity transfer steps:
    a. Convert stylized RGB→ grayscale according to Rec.601 luma (0.299, 0.587, 0.114)
    b. Convert stylized grayscale into YUV (YCbCr)
    c. Convert original image into YUV (YCbCr)
    d. Recombine (stylizedYUV.Y, originalYUV.U, originalYUV.V)
    e. Convert recombined image from YUV back to RGB
13. Save image for specific iteration in numpy array ,and then save the image in path with :

```
    def imsave(path, img):
        img = np.clip(img, 0, 255).astype(np.uint8)
        Image.fromarray(img).save(path, quality=95)
```

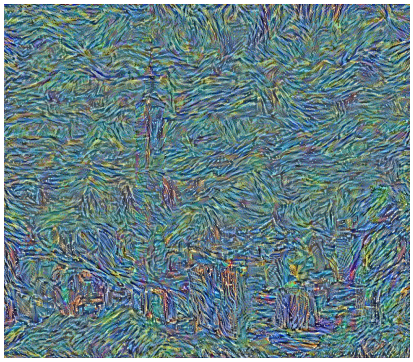
RESULTS



```
Iteration  973/1000 (6 hr 5 min elapsed, 10 min 27 sec remaining)
Iteration  974/1000 (6 hr 6 min elapsed, 10 min 5 sec remaining)
Iteration  975/1000 (6 hr 6 min elapsed, 9 min 43 sec remaining)
Iteration  976/1000 (6 hr 7 min elapsed, 9 min 20 sec remaining)
Iteration  977/1000 (6 hr 7 min elapsed, 8 min 58 sec remaining)
Iteration  978/1000 (6 hr 7 min elapsed, 8 min 36 sec remaining)
Iteration  979/1000 (6 hr 8 min elapsed, 8 min 13 sec remaining)
Iteration  980/1000 (6 hr 8 min elapsed, 7 min 51 sec remaining)
Iteration  981/1000 (6 hr 8 min elapsed, 7 min 29 sec remaining)
Iteration  982/1000 (6 hr 9 min elapsed, 7 min 5 sec remaining)
Iteration  983/1000 (6 hr 9 min elapsed, 6 min 42 sec remaining)
Iteration  984/1000 (6 hr 10 min elapsed, 6 min 20 sec remaining)
Iteration  985/1000 (6 hr 10 min elapsed, 5 min 57 sec remaining)
Iteration  986/1000 (6 hr 10 min elapsed, 5 min 35 sec remaining)
Iteration  987/1000 (6 hr 11 min elapsed, 5 min 13 sec remaining)
Iteration  988/1000 (6 hr 11 min elapsed, 4 min 50 sec remaining)
Iteration  989/1000 (6 hr 11 min elapsed, 4 min 28 sec remaining)
Iteration  990/1000 (6 hr 12 min elapsed, 4 min 6 sec remaining)
Iteration  991/1000 (6 hr 12 min elapsed, 3 min 43 sec remaining)
Iteration  992/1000 (6 hr 12 min elapsed, 3 min 21 sec remaining)
Iteration  993/1000 (6 hr 13 min elapsed, 2 min 59 sec remaining)
Iteration  994/1000 (6 hr 13 min elapsed, 2 min 36 sec remaining)
Iteration  995/1000 (6 hr 14 min elapsed, 2 min 14 sec remaining)
Iteration  996/1000 (6 hr 14 min elapsed, 1 min 52 sec remaining)
Iteration  997/1000 (6 hr 14 min elapsed, 1 min 29 sec remaining)
Iteration  998/1000 (6 hr 15 min elapsed, 1 min 7 sec remaining)
Iteration  999/1000 (6 hr 15 min elapsed, 44 sec remaining)
Iteration 1000/1000 (6 hr 15 min elapsed, 22 sec remaining)
content loss: 1.35043e+06
  style loss: 281951
     tv loss: 142900
  total loss: 1.77528e+06
1000 OrderedDict([('content', 1350427.2), ('style', 281951.1), ('tv', 142900.0), ('total', 1775278.4)])
```

Images generated at each 100 iterations

Iteration - 0 **Iteration - 100**



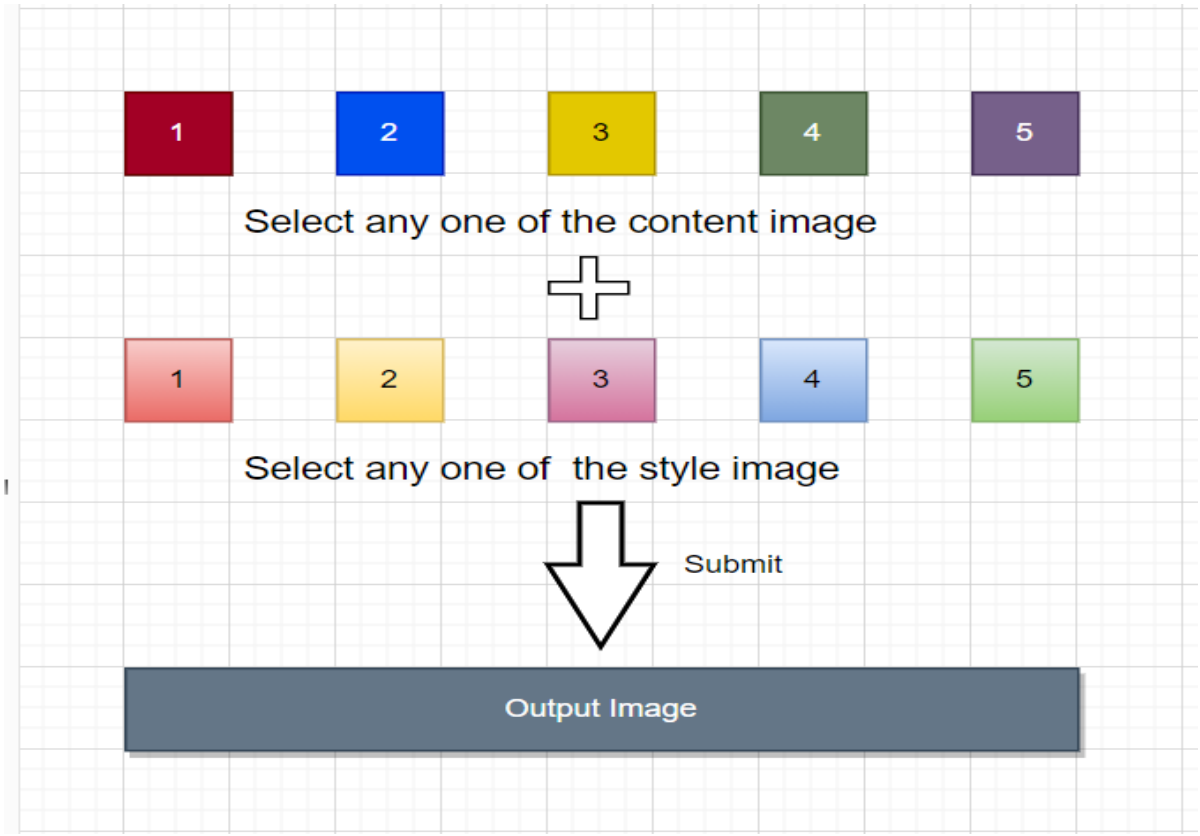
LIMITATIONS OF PROPOSED WORK

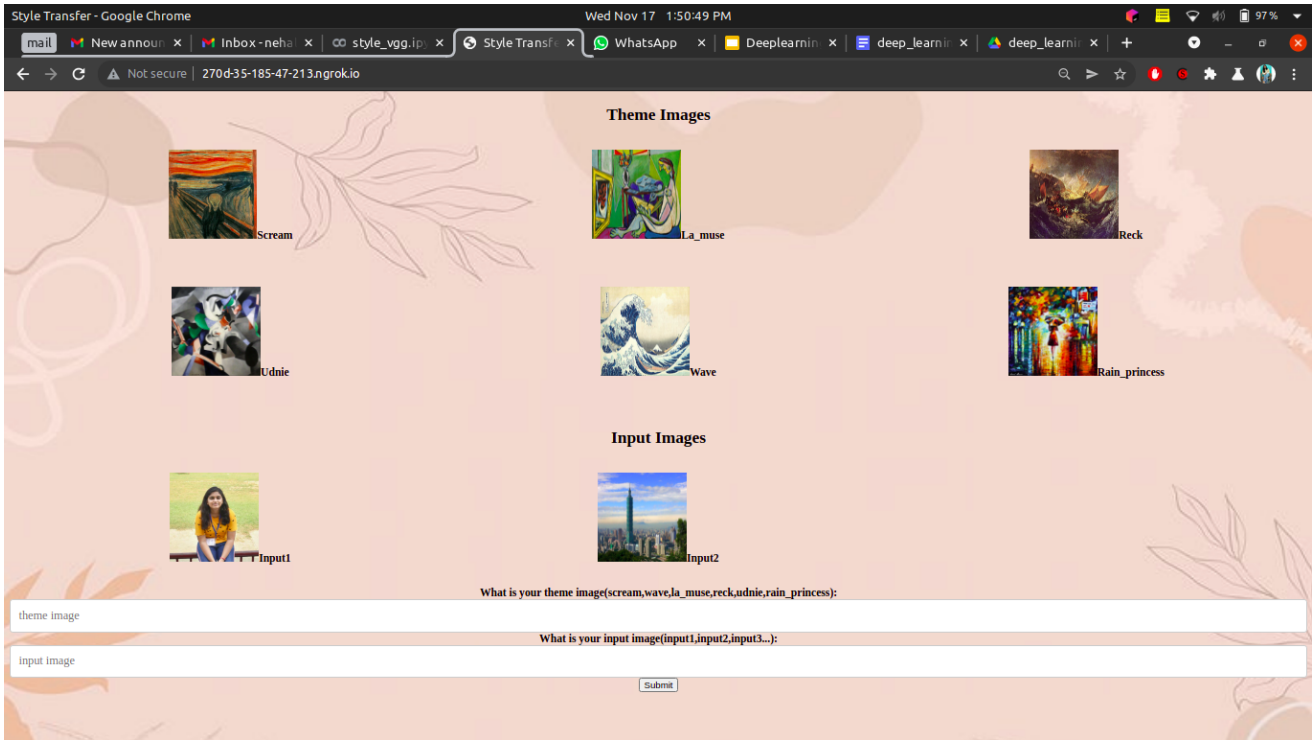
Since convolutional neural networks are typically used for image-classification, we are generally dealing with high-dimensional data (images). We generally need a large amount of data for a convolutional neural network to work effectively.
It takes a very long time to train a convolutional neural network, especially with large datasets. In order to speed up the process we need a specific type of hardware such as a GPU.
CNNs are generally bad at handling rotation and scale-invariance without explicit data augmentation.
High computational cost.

CONCLUSION

In this project we will generate a mixed image out of two images provided, i.e a content image and a style image using CNN with VGG19 model. Here, a mixed image contains style (such as texture, colour) from style image and content from content image. For showing this project we built a web application ,which would allow users to select themes and input images and see their output. It was made using Flask,CSS & HTML. We also calculate the content and style loss for computing the final loss that is 1.77528e+06.

REFERENCES


1
2
3
4
5
Select any one of the content image
1
2
3
4
5
Select any one of the style image
Submit
Output Image


Style Transfer - Google Chrome
Wed Nov 17 1:50:49 PM
Not secure
270d-35-185-47-213.ngrok.io
Theme Images
Scream
La_muse
Reck
Udnie
Wave
Rain_princess
Input Images
Input1
Input2
What is your theme image(scream,wave,la_muse,reck,udnie,rain_princess):
theme image
What is your input image(input1,input2,input3...):
input image
Submit