# Project report

# Title of project: The Image Cartoonifier

Aim:To convert a given input image to another image having a cartoonish style.

Method/the process :

The key concept behind this project is convolution neural networks.Given 2 images ,let us say image 1 is the style image and image 2 is the input image , the input image can be converted using CNN as a base,into another image having the same style as the style image.This is called as Neurtal Style Transfer.

The coding language to be used is Python.

None of this can be done without basics.From week 1 I started building on basics ,learning from the resources given by the project mentors.

Week 1 : basics of python,pandas and numpy

These are two essential libraries for handling with data and making computation faster.

Assignment for week 1 : [SoC/SoCWEEK1.zip at main · may-06/SoC (github.com)](https://github.com/may-06/SoC/blob/main/SoCWEEK1.zip)

Week 2 and 3: Introduction to Machine Learning

Supervised Learning concepts: Linear Regression, Logistic Regression, Regularisation

We were given a practical overview as well.Coding exercises were given as a part of one of the resources which helped in understanding the idea of training a machine to interpret data to predict the output.

Assignment for week 2: [SoC/CORRECTED\_WEEK2 ASSIGNMENT.zip at main · may-06/SoC (github.com)](https://github.com/may-06/SoC/blob/main/CORRECTED_WEEK2%20ASSIGNMENT.zip)

Week 4 and 5:

Neural Networks

Convolutional Neural Newtorks

The topics covered constitute the main core of the project.Playlist links and summary links were given.

It also included the working of computer vision which involves image processing using CNN.

Assignment : [SoC/mod\_assi.py at main · may-06/SoC (github.com)](https://github.com/may-06/SoC/blob/main/mod_assi.py)

Given a database of images (pixel values and the image labels( Each image was a number)),a deep neural network was trained to predict the output(what the image rep.) for a test database consisting of pixel numbers.

Week 7 and 8 :

Handling big data with just numpy and pandas was pretty intensive(as you can see the prev assignment code).PyTorch had libraries to handle data and design neural networks in an efficient way.These 2 weeks dealt with PyTorch .No assignment was given

Week 10:

A paper on Neural Style Transfer was shared.The paper had a detailed explanation of how exactly Neural Style Transfer works and the methods/strategies involved .

I will soon start coding for the final project using all the concepts I’ve learned so far.

# PROJECT:

Libraries used:

* Pytorch
* Torch.nn
* Torch.optim
* PIL
* Matplot
* Torchvision.transforms
* Torchvision.models
* Torchvision.utils

There are 8 main parts in this project:

**-Loading images**

-Images are converted into tensors using the torchvision.transforms library

-First the image is resized into a square image of dim imsize x imsize

-The following function does the same and returns the image to that part of the device which will

hold the image(either cpu or gpu)

-gpu is preferred as it makes computation faster

def image\_loader(path):

img = Image.open(path)

img = transforms.Resize((imsize,imsize))(img)

img = transforms.ToTensor()(img)

return img.to(dev, torch.float)

--Cla

-**Loading the model:**

For neural style transfer a pre-trained VGG-19 newtork is used.The feature maps are extracted from layers 0,5,10,19,28(as advised in the research paper).These layers are convolutional layers due to which the size of the image decreases.

For calculating content loss only the features extracted as output of 19th layer is taken(a layer not too deep nor too shallow in the network must be considered while working with content loss).

**Normalization** : (optional-to make computation faster)

The image tensor is normalized using 2 tensors,a mean tensor and a standard deviation tensor.The tensors used in the code are pre-determined tensors .They were found to work best with vgg networks.

Normalization was implemented by adding a layer to the vgg model at the very beginning to normalize it.

A model is created with normalization as the first layer.Layer of vgg network are added subsequently in the

VGG class under the init function.

class Normalization(nn.Module):

def \_\_init\_\_(self):

super(Normalization, self).\_\_init\_\_()

self.mean = torch.tensor([0.485, 0.456, 0.406]).view(-1,1,1)#works well for vgg networks

self.std = torch.tensor([0.229, 0.224, 0.225]).view(-1,1,1)#works well for vgg networks

def forward(self, img):

return (img - self.mean) / self.std

normalization = Normalization()

model = nn.Sequential(normalization)

vgg=models.vgg19(pretrained=True).features

class VGG(nn.Module):

def \_\_init\_\_(self,chosen\_layers):

super(VGG, self).\_\_init\_\_()

self.f\_layers=chosen\_layers

for i, layer in enumerate(vgg):

model.add\_module(str(i+2),layer)

self.model=model

def forward(self, inp):

features = []

for i, layer in enumerate(self.model):

inp = layer(inp)

if i in self.f\_layers:

features.append(inp)

return features

- The output is initialized as the content image(results are better this way)

**-Calculating losses:**

There are 2 loss compents .

-Content loss

-Content and output images are passed through the model and the features from layer 19 are obtained.Mean Squared loss is computed.

-Style loss

-Style and output images are passed through the network and gram matrices of layers 0,5,10,19,28 are obtained.Sum(over the layers) of Mean squared error is calculated.

Note:Here .detach() is a function used to remove a specified object from the computational graph,so that the program takes lesser time to run.

def Content\_loss():

cont\_loss = 0

content\_features = model\_cont(im\_cont)

for i in content\_features:

i.detach()

output\_features = model\_cont(im\_out)

for content\_feat, output\_feat in zip(content\_features, output\_features):

cont\_loss += torch.mean((content\_feat - output\_feat).detach()\*\*2)

return cont\_loss

def gram\_matrix(input\_layer):

ch, h, w = input\_layer.shape

new = input\_layer.view(ch, h\*w)

gram\_m = torch.mm(new, new.t())

return gram\_m.div(ch\*h\*w)

def Style\_loss():

style\_loss = 0

style\_features = model\_style(im\_style)

for i in style\_features:

i.detach()

output\_features = model\_style(im\_out)

for style\_feat, output\_feat in zip(style\_features, output\_features):

style\_gram = gram\_matrix(style\_feat).detach()

out\_gram = gram\_matrix(output\_feat)

style\_loss += torch.mean((style\_gram - out\_gram)\*\*2)

return style\_loss

-**Gradient descent on total loss:**

This is achieved using optimizers.Adam optimizer was used in this project as it is fast compared to other optimizers .Learning rate was set to 0.004. The parameters in neural style transfer are the pixel values of the output image,which are varied to reduce loss based on gradient descent algorithm

A very high weightage is given to the style loss when compared to content loss,because the ouput image was initialized to the content image.

total\_loss = alpha \* content\_loss + beta \* style\_loss

here alpha=1 and beta =10^6

-In the research paper referred beta/alpha was given as 10^3 but in that paper output image was initialized as a noisy image.

-Number of iterations is set as 1000 because not much of a change was observed in iterations there on.

iterations=500

beta=100000

alpha=1

optimizer=opt.Adam([im\_out],lr=0.003)

for i in range(1,iterations-1):

optimizer.zero\_grad()

style\_loss = Style\_loss()

content\_loss = Content\_loss()

total\_loss = beta \* content\_loss + alpha \* style\_loss

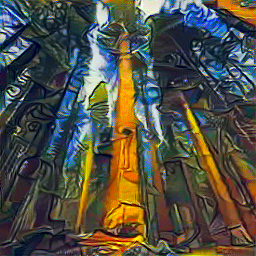
total\_loss.backward()

optimizer.step()

if i%200==0:

save\_image(im\_out,"output{}.png".format(i))

RESULTS:

 A tall tree in a forest

Description automatically generated

+ =

A painting of a person with a hat

Description automatically generated

Image links:

Content image: https://www.britannica.com/plant/tree

Style image: https://artuk.org/discover/artists/picasso-pablo-18811973

A person with long hair

Description automatically generated =

+



Image links:

Style image : <https://in.pinterest.com/pin/513480795011825329/>