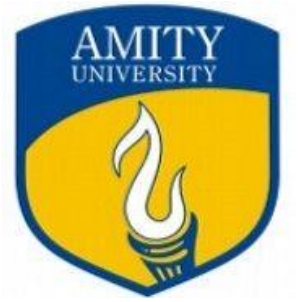
B. Tech – CSE - 3 Continent, 2016-2020

Amity School of Engineering and Technology, Amity University,
Noida, Uttar Pradesh



NEURAL STYLE TRANSFER USING DEEP LEARNING

Style Transferring Photos as well as Real-Time Transfer employing Webcam

By

Sahil Makwane, A2372016030

Utkarsh Ruchir, A2372016031

Minor Project – 7th Semester

Abstract— **Humans - over centuries - have become proficient in the skill of fabricating unique visual stimulations employing the vivid usability of the motley of complex interplay techniques for rendering content based on style of an image. There have been a few algorithms using Artificial Intelligence, Deep Learning Neural Networks to be precise, to generate art. There also have quite a number of publications examining the means of generating art "artificially". In this report, we shall devise a Neural Style Transfer algorithm based on already existing models, having perused 3 notable publications. The style transfer applied on images would manifest as a proof of concept while implementing the same technique on live video by utilizing the webcam is the end goal.**

Keywords – CNN, Deep Learning, Neural Style Transfer, Artificial Intelligence

I. INTRODUCTION

The inception of art lead way to a skill honed in humankind over the course of centuries. The limitation to the variegation in art has been the depths of human imagination. With the advancement of technology disseminating its way – via means of Artificial Intelligence – into several unconventional applications has had a big impact on how we perceive the next step, or perhaps augmentation, of mediums like art.

In our project, our end goal is the style transfer of real-time video employing utilities such as OpenCV, Python and deep learning.

What is Style Transfer (or Neural Style Transfer)? Neural style transfer is an optimization technique used to take two images, an input image and a style image (such as an artwork by a famous painter), — and blend them together such that the input image is transformed to look like the style image.

II. NST ALGORITHMS

The initiation of Neural style transfer algorithms can be cited to Gatys et al. in their 2015 paper titled "A Neural Algorithm of Artistic Style". The working of the neural style transfer is as follows:

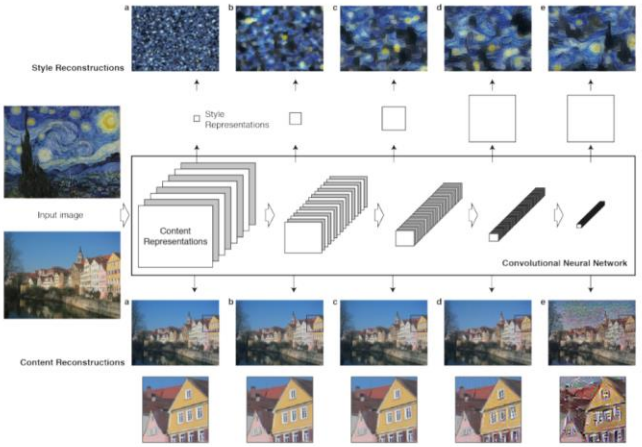


Figure 1: working

The paper by Gatys et al. postulated a NST algorithm which did not enable the need of a different architecture at all but rather only a pre-trained network (like ImageNet) and the requirement to define a loss function which helps output a style transferred image and then optimizes the loss function with every epoch or so.

Hence the question rose was not about which Neural Net to use but rather what loss function to be use in order to address:

1. Content loss,
2. Style Loss and
3. Total - Variation Loss.

All of the above were computed individually and combined in a single

‘meta-loss-function’; and by minimizing the this ‘meta-loss-function’, the content, style and total-variation are optimized as a whole.

Although the results were appropriate, the procedure in itself took more time than one would imagine. At which point, we look at the paper by Johnson et al. named “Perceptual Losses for Real-Time Style Transfer and Super-Resolution” that sought to enforce a neural style transfer algorithm that was three times faster. The method applied NST based off of a super-resolution-like problem based on perceptual loss functions. The major downside however, despite being faster than Gatys et al.’s proposition, was the limitation of the liberty to select images styling images. But rather, a network is specifically trained with a style for the desired image. Once, trained the neural net, we then apply the same to the content image. Even though, as many styled images can be trained to be applied on the content image, the holistic duration of the same cannot be timed upon.

Lastly, we looked at Ulyanov et al.’s publication titled “Instance Normalization: The Missing Ingredient for Fast Stylization”. Their method lead to the fastest real-time performance and probably the best results by utilizing a technique which swaps batch-normalization for instance-normalization; and applying the latter at both the training as well as the testing. Our method has been assisted by all the 3 methods mentioned.

III. WORKING

We commence by apply the style transfer to images and the steps for the same are as follows:

1. Firstly, the required packages are imported and command line arguments are parsed where two command lines required are: a. –model: the model path and b. –image: the input image on which the style needs to be applied. The command line arguments are traversed at the runtime.

```
# importing necessary packages
import argparse
import imutils
import time
import cv2

# construct the argument parser and parse the command line arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="neural style transfer model")
ap.add_argument("-i", "--image", required=True,
    help="input image to apply neural style transfer to")
args = vars(ap.parse_args())
```

Figure 2: Code for Step 1

2. Now, the image is loaded as well as the model and NST is computed. A pre-trained neural style transfer model is loaded into the memory as net; followed by the loading and resizing of the input image; post which a *blob* is constructed; and lastly a forward-pass is performed to obtain an output image.

```
# Here, the neural style transfer model is being loaded from the directory
print("[INFO] style transfer model is being loaded...")
net = cv2.dnn.readNetFromTorch(args["model"])

# input image is anow loaded and resized to a width of 600px, and
# then image dimensions are grapped
image = cv2.imread(args["image"])
image = imutils.resize(image, width=600)
(h, w) = image.shape[:2]

# construct a blob from the image, set the input, and then perform a
# forward pass of the network
blob = cv2.dnn.blobFromImage(image, 1.0, (w, h),
    (103.939, 116.779, 123.680), swapRB=False, crop=False)
net.setInput(blob)
start = time.time()
output = net.forward()
end = time.time()
```

Figure 3: Code for Step 2

3. This step is a salient one. We perform post processing to the output image.

```
# reshape the output tensor, add back in the mean subtraction, and
# then swap the channel ordering
output = output.reshape((3, output.shape[2], output.shape[3]))
output[0] += 103.939
output[1] += 116.779
output[2] += 123.680
output /= 255.0
output = output.transpose(1, 2, 0)
```

Figure 4: Code for Step 3

4. Lastly, we display the image having performed Style Transfer.

```
# show information on how long inference took
print("[INFO] neural style transfer took {:.4f} seconds".format(
    end - start))

# show the images
cv2.imshow("Input", image)
cv2.imshow("Output", output)
cv2.waitKey(0)
```

Figure 5: Code for Step 4

The following structure of command needs to be passed in the terminal to gather the results

```
PS C:\Users\Sahil Makwane\Downloads\neural-style-transfer\neural-style-transfer> python neural_st
le_transfer.py --image images/me.jpg --model models/eccv16/the_wave.t7
[INFO] loading style transfer model...
[INFO] neural style transfer took 3.4847 seconds
```

Figure 6: Command execution in Terminal and output

In the terminal output, the time elapsed to compute the output image is shown – each CNN model is different and the timing for each model varies.

Figure 7: Output after Neural Style Transfer

Having established the working of our method on images, we move onto working towards our end goal of using the style transfer on webcam video as input for real-time style transfer. The procedure involved is quite similar to as we performed the same on static images. Here we:

a) Make use of a special python iterator that allows the cycling of models from our model path.

b) Initialize the webcam stream wherein the frames coming in from the webcam shall be processed in real time or near real time.

c) Loop about the input frames.

d) Execute NST on the frames, do post-processing over the output and present the output on the screen.

e) Iterate over the models at the press of the key “n” to put the iterator to use by cycling models without having to stop the script.

The steps involved are as follows:

1. To begin with, we import the necessary packages/models. At this point, we only require the path to the models. The command line argument, --model, paired with *argparse*, allows us to pass the at runtime itself.

```
# import the necessary packages
from imutils.video import VideoStream
from imutils import paths
import itertools
import argparse
import imutils
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--models", required=True,
    help="path to directory containing neural style transfer models")
args = vars(ap.parse_args())
```

Figure 8: Code for Real-Time NST Step 1

2. To cycle through the different models, we create a model path iterator. Once the frames are under the loop, pressing “n” would switch the models allowing us to perceive

the different models without stopping the script or having to restart it. To construct the iterator, we grab and sort paths to all NST models; give them a unique ID; and user *itertools* and *cycle* to fabricate an iterator. Here, *cycle* permits us to make a circular list which when ends, starts back at the beginning.

```
# grab the paths to all neural style transfer models in our 'models'
# directory, provided all models end with the '.t7' file extension
modelPaths = paths.list_files(args["models"], validExts=(".t7",))
modelPaths = sorted(list(modelPaths))

# generate unique IDs for each of the model paths, then combine the
# two lists together
models = list(zip(range(0, len(modelPaths)), (modelPaths)))

# use the cycle function of itertools that can loop over all model
# paths, and then when the end is reached, restart again
modelIter = itertools.cycle(models)
(modelID, modelPath) = next(modelIter)
```

Figure 9: Code for NST Real-Time Step 2

3. As our first NST is loaded, and video stream initialized, we read the first NST model using its path and grab the frames from the webcam. Now, its time to loop the frames.

```
# load the neural style transfer model from disk
print("[INFO] style transfer model is being loaded, hold on...")
net = cv2.dnn.readNetFromTorch(modelPath)

# initialize the video stream, then allow the camera sensor to warm u
print("[INFO] video stream live in 3..2..1...Kachow!")
vs = VideoStream(src=0).start()
time.sleep(2.0)
print("[INFO] {}. {}".format(modelID + 1, modelPath))
```

Figure 10: Code for NST Real-Time Step 3

4. Here, we initialize the looping of the frames by grabbing the frame, pre-processing it into a *blob* and sending it through the Convolutional Neural Network.

```
# loop over frames from the video file stream
while True:
    # grab the frame from the threaded video stream
    frame = vs.read()

    # resize the frame to have a width of 600 pixels (while
    # maintaining the aspect ratio), and then grab the image
    # dimensions
    frame = imutils.resize(frame, width=600)
    orig = frame.copy()
    (h, w) = frame.shape[:2]

    # construct a blob from the frame, set the input, and then perform a
    # forward pass of the network
    blob = cv2.dnn.blobFromImage(frame, 1.0, (w, h),
        (103.939, 116.779, 123.680), swapRB=False, crop=False)
    net.setInput(blob)
    output = net.forward()
```

Figure 11: Code for NST Real-Time Step 4

5. Now, we shall post process and display the output image. Our output image is “de-process” by reshaping, mean addition, rescaling and transposing. The output is display of both the original as well as processed frames.

```
    # reshape the output tensor, add back in the mean subtraction, and
    # then swap the channel ordering
    output = output.reshape((3, output.shape[2], output.shape[3]))
    output[0] += 103.939
    output[1] += 116.779
    output[2] += 123.680
    output /= 255.0
    output = output.transpose(1, 2, 0)

    # show the original frame along with the output neural style
    # transfer
    cv2.imshow("Input", frame)
    cv2.imshow("Output", output)
    key = cv2.waitKey(1) & 0xFF
```

Figure 12:Code for NST Real-Time Step 5

6. Lastly, we process the key captures. ‘n’: grabs the next NST model path and ID; loads it. The iterator helps cycle back to the beginning. ‘q’: quits the loops.

```
    # if the `n` key is pressed (for "next"), load the next neural
    # style transfer model
    if key == ord("n"):
        # grab the next neural style transfer model model and load it
        (modelID, modelPath) = next(modelIter)
        print("[INFO] {}. {}".format(modelID + 1, modelPath))
        net = cv2.dnn.readNetFromTorch(modelPath)

    # otheriwse, if the `q` key was pressed, break from the loop
    elif key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Figure 13: Code for NST Real-Time Step 6

The output for the real-time NST are shown below.

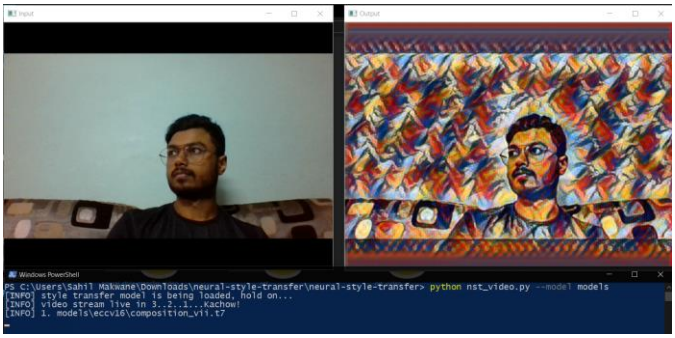


Figure 14: Output Real-Time NST

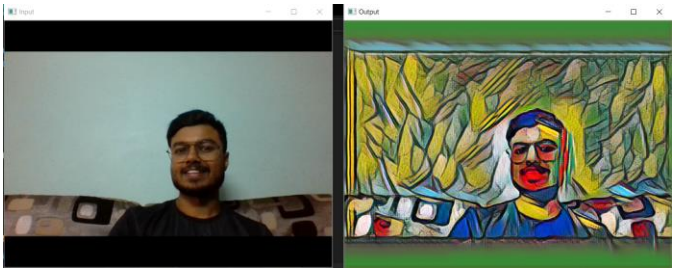


Figure 15: Output after pressing 'n'

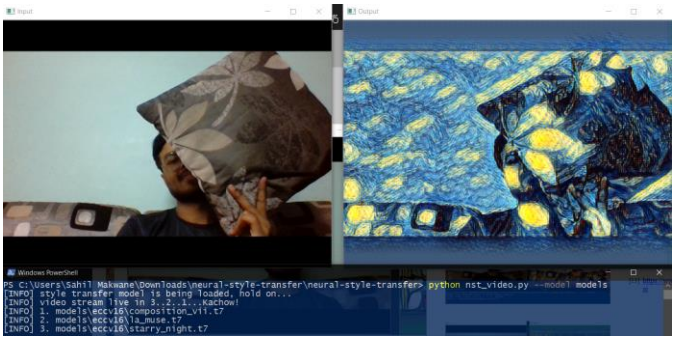


Figure 16: Output after pressing 'n' again

IV. REFERENCES

[1] https://arxiv.org/abs/1508.06576

[2] https://cs.stanford.edu/people/jcjohns/eccv16/

[3] https://arxiv.org/pdf/1607.08022.pdf

[4] https://www.youtube.com/watch?v=Uxax5EKg0zA&t=3s

[5] https://en.wikipedia.org/wiki/Neural_Style_Transfer