ANALYSIS OF DEEP-ART TECHNIQUES AND APPLICATIONS

A PROJECT REPORT

Submitted by

ROSHAN PRIYADARSHI (RA2111026010139)
YASHOWARDHAN SAMDHANI (RA2111026010151)
MEHAK AGARWAL (RA2111026010152)
ANANYA GAUTAM (RA2111026010211)

Under the Guidance of

DR T R SARAVANAN

Associate Professor, Department of Computational Intelligence

In partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

With a specialisation in ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603 203

NOVEMBER 2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**ANALYSIS OF DEEP-ART TECHNIQUES AND APPLICATIONS**" is the bonafide work of Mr Roshan Priyadarshi (RA2111026010139), Mr Yashowardhan Samdhani (RA2111026010151), Ms Mehak Agarwal (RA2111026010152), and Ms Ananya Gautam (RA2111026010211), who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR T R SARAVANAN
Associate Professor,
Department of Computational Intelligence,
SRM Institute of Science and Technology, Kattaknkulathur

DR R ANNIE UTHRA
Professor and Head,
Department of Computational Intelligence,
SRM Institute of Science and Technology, Kattankulathur

Department of Computational Intelligence

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course : B.Tech in Computer Science and Engineering with a specialisation in Artificial Intelligence and Machine Learning

Student Names : Roshan Priyadarshi, Yashowardhan Samdhani, Mehak Agarwal, Ananya Gautam

Registration Number : RA2111026010139, RA2111026010151, RA2111026010152, RA2111026010211

Title of Work : Analysis of Deep-Art Techniques and Applications

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references/listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own

- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T V Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr R Annie Uthra**, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisors, **Dr Saranya A**, Assistant Professor, and **Dr Anitha D**, Assistant Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr T R Saravanan**, Associate Professor, Department of Computational Intelligence, SRM Institute of Science

and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Computational Intelligence Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Mr Roshan Priyadarshi (RA2111026010139)

Mr Yashowardhan Samdhani (RA2111026010151)

Ms Mehak Agarwal (RA2111026010152),

Ms Ananya Gautam (RA2111026010211)

TABLE OF CONTENTS

ABSTRACT

Deep learning and neural networks have witnessed remarkable advancements in recent years, leading to groundbreaking applications across various domains. This research project delves into "Deep-Art" techniques, focusing on the exploration and practical applications of the Neural-Style algorithm developed by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.

Deep-Art techniques harness the power of convolutional neural networks (CNNs) to create stunning and innovative artwork, pushing the boundaries of traditional artistic expression. The Neural-Style algorithm, also known as "Neural-Style Transfer," plays a pivotal role in this endeavour by allowing artists and researchers to blend the stylistic qualities of one image with the content of another. This fusion of content and style through deep neural networks opens up new dimensions of artistic creativity.

The research project commences with an in-depth exploration of the underlying principles of the Neural-Style algorithm, elucidating its architecture, training process, and the optimisation methods that drive its artistic transformation capabilities. By understanding the core mechanics, we aim to provide a comprehensive foundation for the subsequent stages of our research.

The project's second phase investigates a wide range of applications for Neural-Style Transfer. These applications extend beyond traditional artwork to encompass diverse areas such as fashion design, interior decoration, and marketing. By customising and applying the Neural-Style algorithm to specific domains, we aim to unveil the immense potential it offers for enhancing creativity and visual aesthetics.

In conclusion, this research project seeks to bridge the gap between deep learning and artistic expression, specifically through the Neural-Style algorithm. By comprehensively exploring the algorithm's architecture and applications, we aim to shed light on the transformative potential it holds for the creative and technological industries.

INTRODUCTION

1.1. General

Art has been an integral part of human culture for millennia, serving as a medium for creative expression, cultural preservation, and storytelling. Over the years, various art forms have evolved, adapting to the changing technological landscape. In the contemporary era, the intersection of technology and art has created an exciting fusion known as "Deep-Art." This field harnesses the power of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) to create innovative and captivating works of art. This introduction sets the stage for a research project that explores the Neural-Style algorithm within the context of Deep-Art.

1.2. Purpose

The primary purpose of this research project is to unravel the potential and applications of the Neural-Style algorithm, a seminal development in Deep Learning, for art and creativity. By doing so, we aim to foster a better understanding of how AI and DL technologies can transform the creative landscape, bridging the gap between technology and artistic expression.

1.3. Scope

This project encompasses a comprehensive examination of the Neural-Style algorithm, its architecture, and its transformative capabilities in the realm of Deep-Art. We also aim to explore a wide array of applications, extending beyond traditional art to fields like fashion design, interior decoration, and marketing. Additionally, we will investigate the integration of Deep-Art techniques into emerging technologies, particularly within the context of Computer Vision, virtual reality, and augmented reality, to create interactive and immersive artistic experiences.

1.4. Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence is the overarching field that encompasses the development of computer systems capable of performing tasks that typically require human intelligence, such as reasoning, problem-solving, and learning. These programs and applications imitate the behaviour of the

human mind and, hence, require superabundant knowledge related to all the variables of the problem. These include the various objects, their properties, differentiating categories and the relationships between them.

Machine Learning is a subset of AI that enables machines to learn from data and make predictions or decisions based on that learning. It can be broadly classified into three categories based on the data they use to learn and the type of outputs they produce. They are:

- Supervised Learning: In this type of learning, the datasets that the machine utilises to learn are entirely structured and labelled. The datasets have input features and their corresponding outputs and are given to the ML algorithm as inputs for training. While training, the model learns by mapping its predictions to the true predictions present in the dataset, evaluating its performance, and automatically updating its learning curve to achieve better scores of the performance metrics. For example, classification and regression tasks such as image and commodity price prediction, respectively.
- Unsupervised Learning: In this type of learning, the datasets are not mapped with their outputs. The models are trained on a dataset of several features and are expected to produce outputs by learning and identifying common patterns without conforming to a certain “correct answer”. For example, clustering algorithms are used for customer segmentation, DNA pattern recognition and grouping in evolutionary biology.
- Reinforcement Learning: In this type of learning, the algorithm updates its learning curve by studying the features of the learning problem, environment and the agent's behaviour to maximise its performance. It works on an action-reward-based system by learning any method well suited to solve the problem while correcting any method that does not. The ultimate goal is reached efficiently and rapidly, and the machine learns only those methods that facilitate this. For example, autonomous driving and gaming technologies.

Deep Learning is a subfield of Machine Learning characterised by using deep neural networks, specifically deep artificial neural networks, to model complex patterns in data. The model called an Artificial Neural Network, is similar to the human brain, where connections exist between the individual units (neurons). However, unlike the human brain, the individual perceptrons are

restricted to forming several connections with certain layers. They are allowed to propagate data only in restricted directions defined according to the type of the neural network.

Neural networks can be designed and trained to perform any task that machine learning algorithms can and are more robust than ML models. These networks have the ability to learn from raw data and grasp the various features as the data is passed on through the different layers while simultaneously compressing the data and producing the output from these sets of features extracted during propagation.

1.5. Computer Vision

Computer Vision is a critical component of this research, providing the foundational technology for many Deep-Art applications. It involves enabling computers to interpret and understand visual information from the world, including images and videos. Within deep art, Computer Vision techniques are essential for processing and manipulating visual content to generate artistic outputs. The seamless integration of Computer Vision with Neural-Style algorithms offers unparalleled opportunities to create visually stunning and conceptually rich artworks that challenge our understanding of creativity and artistic expression.

LITERATURE REVIEW

YEAR	PAPER TITLE	JOURNAL NAME	ALGORITHM /TECHNIQUE USED	DATASET USED	INFERENCE (LIMITATIONS/FUTURE WORK)
2016	"A Neural Algorithm of Artistic Style"	arXiv preprint	Neural style transfer using CNNs	Various images for content and style	Limitations include computational cost and style control. Future work can focus on efficiency and style control improvements.
2019	"DeepDream - a Code Example for Visualizing Neural Networks"	arXiv preprint	DeepDream algorithm for visualising neural networks	Various images for visualisation	DeepDream primarily focuses on neural network visualisation rather than traditional art generation. Future work could involve adapting the algorithm for more artistic applications and enhancing control over the generated images.
2021	"Image-to-Image Translation with Conditional Adversarial Networks"	arXiv preprint	Conditional GANs for image-to-image translation	Various paired image datasets	Limitations include the need for paired data and challenges in translating complex scenes. Future work may involve improving the effectiveness of conditional GANs for diverse image translation tasks and exploring data-efficient approaches.

Neural Style Algorithm

The Neural-Style algorithm, developed by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, is a deep learning-based approach for transferring the artistic style of one image onto the content of another image. This technique combines the content of one image with the style of another to create visually appealing and artistic results. The process involves three main components: a content image, a style image, and a generated image, which is the output.

1. Convolutional Neural Networks (CNNs): The foundation of the Neural-Style algorithm is Convolutional Neural Networks (CNNs). CNNs are deep learning models commonly used for image processing tasks. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. These networks are capable of learning hierarchical features from images.
2. Content Image: The content image is the input image whose content we want to preserve in the final result. The deep neural network processes this image and extracts its content features. In the context of CNNs, content features are usually extracted from the later layers of the network, as these layers capture higher-level image representations.
3. Style Image: The style image is another input image from which we want to extract the artistic style. To capture the style, the algorithm focuses on the correlations between different features and textures within this image. The deeper layers of the CNN are used to capture these correlations.
4. Feature Extraction: The Neural-Style algorithm selects specific layers within a pre-trained CNN, typically a VGG network (a type of CNN), for feature extraction. The choice of layers for content and style extraction can be adjusted to control the balance between content and style in the final result.
5. Content Features: The content features are obtained by passing the content image through the selected network layers. These features represent the high-level content of the image, such as objects, shapes, and structures.
6. Style Features: The style features are extracted by running the style image through the same network layers. These features capture information about the textures, colours, and patterns found in the style image.

7. Loss Function: The Neural-Style algorithm defines a loss function that measures the difference between the generated image and the content and style images. The loss function has two main components:
8. Content Loss: The content loss quantifies how different the content features of the generated image are from those of the content image. This is typically computed as the mean squared error (MSE) between the feature representations in the selected layers of the CNN.
9. Style Loss: The style loss measures the difference between the style features of the generated image and those of the style image. It is calculated using a Gram matrix, which encodes the correlations between different feature maps. The style loss is often represented as the mean squared error between the Gram matrices of the style features.
10. Total Loss: The total loss is a weighted combination of content loss and style loss, and it is used to optimise the generated image. The algorithm aims to find an image that minimises this total loss, effectively balancing the preservation of content and the adoption of the desired style.
11. Optimisation: The Neural-Style algorithm employs an optimisation technique, such as gradient descent, to iteratively update the generated image to minimise the total loss. This involves making small adjustments to the pixel values of the generated image to gradually converge towards a result that matches the content of the content image and exhibits the artistic style of the style image.
12. Iterative Process: The optimisation process is repeated for a certain number of iterations, with the generated image evolving at each step. The final generated image represents the result of the Neural-Style algorithm, blending the content of the content with the style of the style image.

PROPOSED METHODOLOGY

Underlying Principle

The principle is simple: we define two distances, one for the content (D_C) and one for the style (D_S). D_C measures how different the content is between two images, while D_S measures how different the style is between two images. Then, we take a third image, the input, and transform it to minimise its content-distance with the content-image and style-distance with the style-image.

Loss Function

Content Loss

The content loss is a function that represents a weighted version of the content distance for an individual layer. The function takes the feature maps F_{XL} of layer L in a network processing input X and returns the weighted content distance $w_{CL}.D_C^L(X, C)$ between the image X and the content image C. The function must know the feature maps of the content image (F_{CL}) to calculate the content distance. We implement this function as a torch module with a constructor that takes F_{CL} as an input. The distance $||F_{XL} - F_{CL}||^2$ is the mean square error between the two sets of feature maps and can be computed using nn.MSELoss.

We will add this content loss module directly after the convolution layer(s) used to compute the content distance. This way, each time the network is fed an input image, the content losses will be computed at the desired layers, and because of auto-grad, all the gradients will be computed. Now, to make the content loss layer transparent, we must define a forward method that computes the content loss and then returns the layer's input. The computed loss is saved as a parameter of the module.

Style Loss

The style loss module is implemented similarly to the content loss module. It acts as a transparent layer in a network, computing the style loss of that layer. To calculate the style loss, we first need to compute the Gram matrix G_{XL}. The Gram matrix is obtained by multiplying a

given matrix by its transposed matrix. In this application, the given matrix is a reshaped version of the feature maps F_{XL} of layer L. F_{XL} is reshaped to form the matrix F'_{XL}, which has dimensions KxN, where K is the number of feature maps at layer L, and N is the length of any vectorised feature map F_{XL}^{k}. For example, the first line of F'_{XL} corresponds to the first vectorised feature map F_{XL}^{1}.

Finally, the Gram matrix must be normalised by dividing each element by the total number of elements in the matrix. This normalisation is essential to counteract that F'_{XL} matrices with a large N dimension yield larger values in the Gram matrix. These larger values can cause the first layers (before pooling layers) to have a larger impact during the gradient descent. Style features tend to be in the deeper layers of the network, so this normalisation step is crucial.

Model

Basic Structure:
VGG19 comprises 19 layers, including 16 convolutional and 3 fully connected layers. The key architectural features of VGG19 are the repeated stacks of convolutional layers and pooling layers, which make it a deep and effective feature extractor.

Convolutional Layers:
VGG19 primarily employs 3x3 convolutional filters with a stride of 1 and a padding of 1 (which preserves spatial dimensions). These filters extract various image features such as edges, textures, and object parts. The convolutional layers are organised into five blocks, each with multiple convolutional layers.

Pooling Layers:
After each convolutional block, VGG19 includes max-pooling layers with 2x2 pooling windows and a stride of 2. Max-pooling reduces the spatial dimensions of the feature maps, which helps in capturing important information and making the network more computationally efficient.

Fully Connected Layers:

VGG19 ends with three fully connected layers. These layers are used for image classification in the original VGG model. The first two fully connected layers have 4,096 neurons each, and the final layer has as many neurons as the number of classes in the classification task.

Activation Function:

Rectified Linear Units (ReLU) are the activation function after each convolutional and fully connected layer. ReLU activation helps introduce non-linearity and allows the network to learn complex features.

Dropout:

The original VGG model did not include dropout layers, but VGG19, like other variants, may include dropout layers between fully connected layers to prevent overfitting during training.

Output Layer:

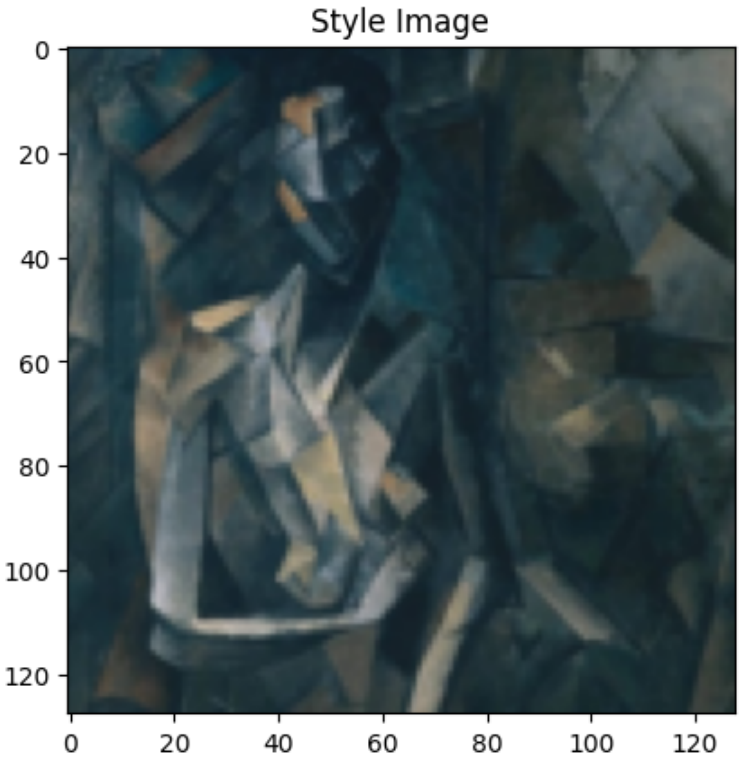
In image classification, the final fully connected layer is followed by a softmax activation function to produce class probabilities.

Pre-Trained Models:

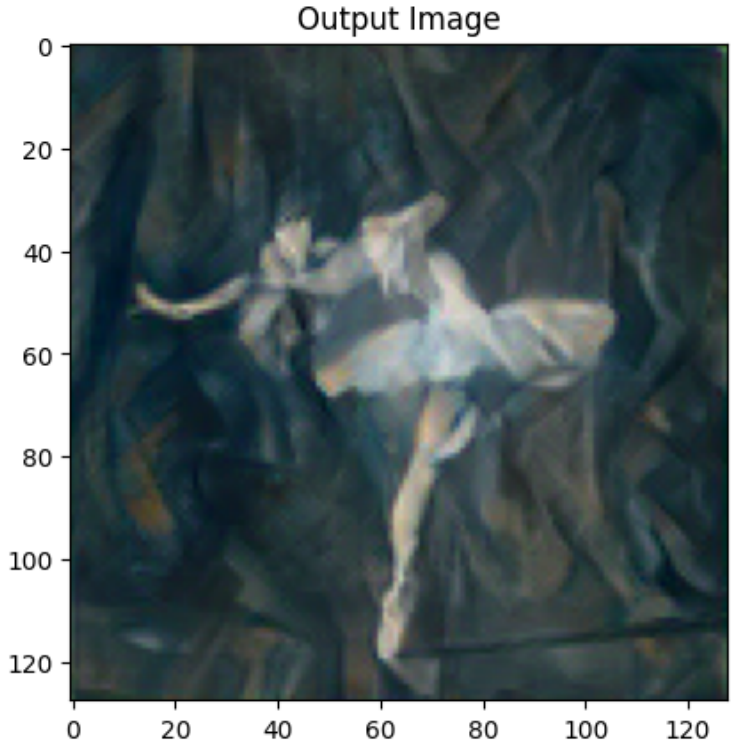
VGG19 is often used as a pre-trained model, especially for transfer learning and feature extraction. Researchers and practitioners have pre-trained VGG19 on large datasets like ImageNet, making it a valuable tool for computer vision tasks.

RESULTS

Inputs



Output



Loss

```
run [50]:
Style Loss : 26.476486 Content Loss: 9.871113

run [100]:
Style Loss : 7.033300 Content Loss: 8.725686

run [150]:
Style Loss : 3.777328 Content Loss: 8.008000

run [200]:
Style Loss : 2.695870 Content Loss: 7.461153

run [250]:
Style Loss : 2.212383 Content Loss: 7.119155

run [300]:
Style Loss : 1.952365 Content Loss: 6.905590
```

CONCLUSION

In conclusion, the study and implementation of the Neural-Style algorithm within the domain of deep art represent a significant stride in the ongoing collaboration between artificial intelligence and creative expression. This project has offered a comprehensive examination of the algorithm's theoretical foundations, demonstrating the intricacies of Convolutional Neural Networks and Gram matrix representations that drive the magic of style transfer.

Through practical implementation and case studies, we have witnessed the algorithm's transformative potential in generating artworks that seamlessly blend the essence of content and style. From image transformations to digital painting and even video processing, the versatility of the Neural-Style algorithm opens new horizons for artists, designers, and creators.

Furthermore, the ethical considerations surrounding deep-art and AI-generated art have been thoughtfully explored. As technology and creativity converge, questions of authorship, copyright, and authenticity emerge as critical discussions within the artistic community. These ethical concerns underscore the need for an ongoing dialogue to ensure the responsible and meaningful use of such powerful tools.

As we move forward, the Neural-Style algorithm stands as a testament to the ever-evolving landscape of creative expression. Its potential for democratising artistic styles and enabling collaboration between human artists and AI systems positions it at the forefront of a new era in artistry. This study is a valuable resource for those looking to harness the power of deep learning in their creative endeavours.

In a world where the boundaries between human creativity and artificial intelligence continue to blur, the Neural-Style algorithm offers a glimpse into a future where innovation knows no bounds. As artists and technologists come together, the fusion of human vision and machine intelligence promises to unlock uncharted realms of artistic innovation, ultimately redefining how we perceive and create art.

FUTURE SCOPE

The future scope of research and application in deep art, particularly focusing on the Neural-Style algorithm, is promising and exciting. As technology advances and our understanding of artificial intelligence deepens, here are some key areas that hold significant potential for exploration:

1. Advanced Style Transfer Techniques: The development of more advanced and sophisticated style transfer techniques that surpass the current capabilities of the Neural-Style algorithm is a promising avenue. Researchers can work on refining the algorithms to allow for more fine-grained control over artistic style transfer, potentially enabling artists to create highly customised and intricate results.
2. Interactive and Real-time Applications: Integrating deep-art techniques into real-time applications, such as interactive installations, mobile apps, or augmented reality (AR) experiences, holds immense potential. This will enable users to engage with AI-powered art in novel and dynamic ways, bridging the gap between traditional art forms and emerging technologies.
3. Multimodal Deep-Art: Exploring the fusion of various modalities, including visual art, music, and text, using deep learning techniques. This interdisciplinary approach can create multisensory art experiences, where AI generates art that combines visual, auditory, and textual elements cohesively.
4. Ethical and Legal Frameworks: As AI-generated art becomes more prevalent, it is crucial to establish ethical and legal frameworks governing copyright, attribution, and authenticity. Future research can focus on creating guidelines and policies to ensure fairness, transparency, and accountability in AI-generated art.
5. Human-AI Collaboration: Investigating how human artists and AI systems can collaborate to push the boundaries of creativity. This collaborative approach may lead to developing novel artistic genres, styles, and techniques that are unattainable through human or AI efforts alone.
6. Artificial General Intelligence (AGI) Integration: As AGI systems become more advanced, there is potential for integrating such systems into the creative process. Future

research may explore how AGI can understand and interpret human artistic intentions and provide insights or suggestions for enhancing the creative process.

7. Education and Training: Developing educational programs and tools to empower artists and designers to harness deep-art techniques effectively. This will enable a broader community of creators to leverage AI for artistic expression.
8. Cross-Domain Applications: Extending deep-art techniques to other fields beyond art, such as scientific visualisation, data interpretation, and healthcare, can unlock new opportunities for problem-solving and innovation.
9. AI Art Curation and Critique: Developing AI systems that can curate and critique AI-generated art, providing valuable feedback to artists and helping to define aesthetic standards in this emerging field.
10. Cultural and Societal Impacts: Ongoing research into the cultural and societal impacts of AI-generated art, including how it influences artistic movements, challenges traditional notions of authorship, and contributes to a broader understanding of the relationship between humans and machines.

The future scope of deep art and the Neural-Style algorithm is a testament to the continuous evolution of AI in creative endeavours. The opportunities for innovation are vast, and as researchers and artists collaborate, the boundaries of what is possible in art will continue to expand, offering fresh perspectives and groundbreaking insights into the intersection of technology and human creativity.

REFERENCES

1. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint.

2. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). A Neural Algorithm of Artistic Style. arXiv preprint.

3. Elgammal, A., Liu, B., Elhoseiny, M., & Mazzone, M. (2017). CAN: Creative Adversarial Networks, Generating" Art" by Learning About Styles and Deviating from Style Norms. arXiv preprint.

4. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image Style Transfer Using Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

5. Alexis Jacq & Winston Herring. Pytorch - Neural Transfer Using Pytorch

APPENDIX

Appendix A: Glossary of Terms

Deep Learning: A subset of machine learning that employs artificial neural networks to model and solve complex problems.

Neural-Style Algorithm: A deep learning technique for transferring the artistic style of one image onto the content of another, as developed by Gatys, Ecker, and Bethge.

Convolutional Neural Network (CNN): A type of deep neural network commonly used for image and video analysis.

Gram Matrix: A mathematical construct that captures style information from feature maps in deep learning.

Feature Extraction: The process of capturing important information or features from data, often used in image analysis and deep learning.

Loss Function: A mathematical function that quantifies the error or difference between predicted and actual values, used to train and optimize machine learning models.

VGG19: A deep convolutional neural network architecture for image classification and feature extraction.

Rectified Linear Unit (ReLU): An activation function that introduces non-linearity by outputting the input directly if it is positive; otherwise, it outputs zero.

Dropout: A regularisation technique used in neural networks to prevent overfitting by randomly deactivating a fraction of neurons during training.

Appendix B: Pre-trained Models

VGG19 Pre-trained Model: The VGG19 model, is pre-trained on large image datasets like ImageNet, and commonly used as a feature extractor in deep learning projects.

Appendix C: Code Samples

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from PIL import Image
import matplotlib.pyplot as plt

import torchvision.transforms as transforms
import torchvision.models as models

import copy

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.set_default_device(device)

# desired size of the output image
imsize = 512 if torch.cuda.is_available() else 128  # use small size if no
GPU

loader = transforms.Compose([
    transforms.Resize(imsize),  # scale imported image
    transforms.ToTensor()])  # transform it into a torch tensor


def image_loader(image_name):
    image = Image.open(image_name)
    # fake batch dimension required to fit network's input dimensions
    image = loader(image).unsqueeze(0)
    return image.to(device, torch.float)


style_img = image_loader("./images/picasso.jpg")
content_img = image_loader("./images/dancing.jpg")

assert style_img.size() == content_img.size(), \
    "we need to import style and content images of the same size"

unloader = transforms.ToPILImage()  # reconvert into PIL image
```

```
plt.ion()

def imshow(tensor, title=None):
    image = tensor.cpu().clone()  # we clone the tensor to not do changes
on it
    image = image.squeeze(0)      # remove the fake batch dimension
    image = unloader(image)
    plt.imshow(image)
    if title is not None:
        plt.title(title)
    plt.pause(0.001) # pause a bit so that plots are updated


plt.figure()
imshow(style_img, title='Style Image')

plt.figure()
imshow(content_img, title='Content Image')

class ContentLoss(nn.Module):

    def __init__(self, target,):
        super(ContentLoss, self).__init__()
        # we 'detach' the target content from the tree used
        # to dynamically compute the gradient: this is a stated value,
        # not a variable. Otherwise the forward method of the criterion
        # will throw an error.
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input

def gram_matrix(input):
    a, b, c, d = input.size()  # a=batch size(=1)
    # b=number of feature maps
    # (c,d)=dimensions of a f. map (N=c*d)

    features = input.view(a * b, c * d)  # resize F_XL into \hat F_XL
```

```
    G = torch.mm(features, features.t())  # compute the gram product

    # we 'normalize' the values of the gram matrix
    # by dividing by the number of element in each feature maps.
    return G.div(a * b * c * d)

class StyleLoss(nn.Module):

    def __init__(self, target_feature):
        super(StyleLoss, self).__init__()
        self.target = gram_matrix(target_feature).detach()

    def forward(self, input):
        G = gram_matrix(input)
        self.loss = F.mse_loss(G, self.target)
        return input

cnn = models.vgg19(pretrained=True).features.eval()

cnn_normalization_mean = torch.tensor([0.485, 0.456, 0.406])
cnn_normalization_std = torch.tensor([0.229, 0.224, 0.225])

# create a module to normalize input image so we can easily put it in a
# ``nn.Sequential``
class Normalization(nn.Module):
    def __init__(self, mean, std):
        super(Normalization, self).__init__()
        # .view the mean and std to make them [C x 1 x 1] so that they can
        # directly work with image Tensor of shape [B x C x H x W].
        # B is batch size. C is number of channels. H is height and W is
width.
        self.mean = torch.tensor(mean).view(-1, 1, 1)
        self.std = torch.tensor(std).view(-1, 1, 1)

    def forward(self, img):
        # normalize ``img``
        return (img - self.mean) / self.std

# desired depth layers to compute style/content losses :
```

```
content_layers_default = ['conv_4']
style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']

def get_style_model_and_losses(cnn, normalization_mean, normalization_std,
                               style_img, content_img,
                               content_layers=content_layers_default,
                               style_layers=style_layers_default):
    # normalization module
    normalization = Normalization(normalization_mean, normalization_std)

    # just in order to have an iterable access to or list of content/style
    # losses
    content_losses = []
    style_losses = []

    # assuming that ``cnn`` is a ``nn.Sequential``, so we make a new
``nn.Sequential``
    # to put in modules that are supposed to be activated sequentially
    model = nn.Sequential(normalization)

    i = 0  # increment every time we see a conv
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = 'conv_{}'.format(i)
        elif isinstance(layer, nn.ReLU):
            name = 'relu_{}'.format(i)
            # The in-place version doesn't play very nicely with the
``ContentLoss``
            # and ``StyleLoss`` we insert below. So we replace with
out-of-place
            # ones here.
            layer = nn.ReLU(inplace=False)
        elif isinstance(layer, nn.MaxPool2d):
            name = 'pool_{}'.format(i)
        elif isinstance(layer, nn.BatchNorm2d):
            name = 'bn_{}'.format(i)
        else:
            raise RuntimeError('Unrecognized layer:
{}'.format(layer.__class__.__name__))
```

```
        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img).detach()
            content_loss = ContentLoss(target)
            model.add_module("content_loss_{}".format(i), content_loss)
            content_losses.append(content_loss)

        if name in style_layers:
            # add style loss:
            target_feature = model(style_img).detach()
            style_loss = StyleLoss(target_feature)
            model.add_module("style_loss_{}".format(i), style_loss)
            style_losses.append(style_loss)

    # now we trim off the layers after the last content and style losses
    for i in range(len(model) - 1, -1, -1):
        if isinstance(model[i], ContentLoss) or isinstance(model[i],
StyleLoss):
            break

    model = model[:(i + 1)]

    return model, style_losses, content_losses

input_img = content_img.clone()
# if you want to use white noise by using the following code:
#
# ::
#
#    input_img = torch.randn(content_img.data.size())

# add the original input image to the figure:
plt.figure()
imshow(input_img, title='Input Image')

def get_input_optimizer(input_img):
    # this line to show that input is a parameter that requires a gradient
```

```
    optimizer = optim.LBFGS([input_img])
    return optimizer

def run_style_transfer(cnn, normalization_mean, normalization_std,
                       content_img, style_img, input_img, num_steps=300,
                       style_weight=1000000, content_weight=1):
    """Run the style transfer."""
    print('Building the style transfer model..')
    model, style_losses, content_losses = get_style_model_and_losses(cnn,
        normalization_mean, normalization_std, style_img, content_img)

    # We want to optimize the input and not the model parameters so we
    # update all the requires_grad fields accordingly
    input_img.requires_grad_(True)
    # We also put the model in evaluation mode, so that specific layers
    # such as dropout or batch normalization layers behave correctly.
    model.eval()
    model.requires_grad_(False)

    optimizer = get_input_optimizer(input_img)

    print('Optimizing..')
    run = [0]
    while run[0] <= num_steps:

        def closure():
            # correct the values of updated input image
            with torch.no_grad():
                input_img.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_img)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.loss
            for cl in content_losses:
                content_score += cl.loss
```

```
            style_score *= style_weight
            content_score *= content_weight

            loss = style_score + content_score
            loss.backward()

            run[0] += 1
            if run[0] % 50 == 0:
                print("run {}:".format(run))
                print('Style Loss : {:4f} Content Loss: {:4f}'.format(
                    style_score.item(), content_score.item()))
                print()

            return style_score + content_score

        optimizer.step(closure)

    # a last correction...
    with torch.no_grad():
        input_img.clamp_(0, 1)

    return input_img

output = run_style_transfer(cnn, cnn_normalization_mean,
cnn_normalization_std,
                            content_img, style_img, input_img)

plt.figure()
imshow(output, title='Output Image')

# sphinx_gallery_thumbnail_number = 4
plt.ioff()
plt.show()
```