Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Exploring the Applicability of Deterministic Decoupling to Convolutional Networks for Style Transfer and Texture Synthesis

Autor(a): Daniel Salvadó Ormad

Tutor(a): Francisco Javier de la Portilla Muelas
Luis Baumela Molina

Madrid, «July 2023»

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Exploring the Applicability of Deterministic Decoupling to Convolutional Networks for Style Transfer and Texture Synthesis

«July 2023»

Autor(a): Daniel Salvadó Ormad
Tutor(a): Francisco Javier de la Portilla Muelas
Luis Baumela Molina
Computer Vision
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El análisis de texturas es un componente fundamental de la visión artificial, con aplicaciones en varios dominios, como el análisis de imágenes y la síntesis de texturas. Esta investigación explora la viabilidad de mejorar el poder descriptivo de las características para el análisis de texturas mediante la combinación de técnicas de desacoplamiento determinista con métodos avanzados para el modelado de texturas. El objetivo principal es investigar el potencial para desacoplar características globales extraídas por una red neuronal convolucional (CNN) para describir texturas con precisión.

Para abordar este objetivo, se ha desarrollado un modelo para extraer características representativas de las texturas utilizando una CNN previamente entrenada. La atención se centra en capturar las medias espaciales de las respuestas de los canales, ya que desempeñan un papel crucial en la representación de características importantes de la textura. El objetivo es mejorar el poder descriptivo de las características extraídas para el análisis y la representación de la textura visual.

Además, se ha diseñado una nueva función de pérdida para calcular la distancia entre texturas. Esta función de pérdida permite una optimización precisa del rendimiento del modelo, proporcionando una medida de similitud o diferencia entre texturas.

Si bien las características seleccionadas no cumplen con los requisitos teóricos para un desacoplamiento perfecto, investigaciones anteriores han demostrado que las técnicas de desacoplamiento determinista aún pueden ser efectivas en escenarios donde no se cumplen estos requisitos. Esta observación motiva la exploración del potencial de las técnicas de desacoplamiento para las CNN.

El modelo propuesto se evalúa exhaustivamente en una variedad de tareas, mostrando su versatilidad y aplicabilidad práctica. Se investiga la viabilidad de desacoplar las características, resaltando las ventajas y limitaciones de la metodología integrada.

A través de esta investigación, se obtienen conocimientos sobre las complejidades del modelado de texturas y las limitaciones de los enfoques existentes. Al contribuir al avance del análisis y la representación de texturas visuales, esta investigación ofrece potencial para mejorar las aplicaciones de visión artificial. Establece una base para una mayor exploración y desarrollo, con el objetivo de obtener resultados más realistas y visualmente cautivadores en el análisis y la síntesis de texturas.

Abstract

Visual texture analysis is a fundamental component of computer vision, with applications in various domains such as image analysis and texture synthesis. This research explores the feasibility of enhancing the descriptive power of features for texture analysis by combining deterministic decoupling techniques with advanced methods for texture modelling. The main objective is to investigate the potential for decoupling global features extracted by a convolutional neural network (CNN) to accurately describe textures.

To address this objective, a model is developed to extract meaningful features from textures using a pre-trained CNN. The focus is on capturing the spatial means of channel responses, as they play a crucial role in representing important texture features. The aim is to enhance the descriptive power of the extracted features for visual texture analysis and representation.

In addition, a novel loss function is designed to compute the distance between textures. This loss function enables precise optimization of the model's performance by providing a measure of similarity or dissimilarity between textures.

While the selected features do not meet the theoretical requirements for perfect decoupling, previous research has shown that deterministic decoupling techniques can still be effective in scenarios where these requirements are not met. This observation motivates the exploration of the potential of decoupling techniques for CNNs.

The proposed model is thoroughly evaluated across a range of tasks, showcasing its versatility and practical applicability. The feasibility of decoupling the features is investigated, shedding light on the advantages and limitations of the integrated methodology.

Through this research, insights are gained into the complexities of texture modelling and the limitations of existing approaches. By contributing to the advancement of visual texture analysis and representation, this research offers potential for improving computer vision applications. It establishes a foundation for further exploration and development, aiming to realize more realistic and visually captivating results in texture analysis and synthesis.

Contents

Chapter 1

Introduction

1.1 Background and Significance

Visual textures are fundamental elements in the perception and aesthetic appreciation of natural images. They encompass regular patterns formed by the arrangement of colours or intensities within an image, playing a pivotal role in computer vision. The comprehension and accurate representation of visual textures carry substantial significance across diverse real-world applications. From image analysis and object recognition to texture synthesis and human perception, the effective understanding and representation of visual textures can have far-reaching implications in various fields. By harnessing the power of visual texture analysis, we can unlock new possibilities for advancing computer vision and enhancing our understanding of the visual world.

While characterizing natural images can be challenging, visual textures lend themselves more readily to mathematical and statistical representation. The pioneering research conducted by Julesz [1] focused on investigating how the human visual system discriminates between different patterns and the role of spatial organization in visual perception. His studies demonstrated that humans can discern various random dot patterns based solely on their spatial organization, without relying on semantic information or object recognition. Julesz's research paved the way for the development of statistical approaches to analyse and describe visual textures. Instead of relying on a pixel-by-pixel memorization of an image, Julesz's statistical representation of textures mirrored the way humans interpret patterns. These techniques leverage statistical processing capabilities to identify patterns and distributions in texture data. This representation of visual textures provides valuable insights into their characteristics and structure, enabling their application in various fields and domains.

Significant progress has been achieved in the domain of visual texture representation. For instance, Geman and Geman made notable contributions by utilizing Markov random fields in their work [2]. In another notable advancement, Portilla and Simoncelli proposed an effective parametric modelling approach that effectively tackles these challenges by leveraging complex wavelet coefficients [3].

In recent years, the emergence of neural networks has revolutionized the field of visual texture representation. These computational models hold great potential for

comprehending and representing intricate patterns, including visual textures. Bethge popularized the use of convolutional neural networks (CNNs) for texture modelling [4], and even achieved successful style transfer with them [5]. Simoncelli demonstrated the effectiveness of employing the spatial means of feature maps from a CNN as a parametric model for visual textures [6].

Moreover, researchers have explored additional avenues for enhancing visual texture analysis. Martínez-Enríquez, González, and Portilla proposed a formalism called nested normalizations, which focuses on decoupling features based on defining transformations on submanifolds [7]. This has been found to improve data analysis performance and feature transfer on textures in regression and classification tasks, even when all the theoretical requirements for exact decoupling were not met.

This study presents a novel approach that combines the power of neural networks with the innovative nested normalizations method for visual texture analysis and representation. By integrating deterministic decoupling, convolutional neural networks, and texture modelling, this research aims to advance our understanding of visual textures, expand the capabilities of computer vision systems, and unlock new possibilities for a wide range of real-world applications. The synergy between these three key elements is visually depicted in Figure 1.1, showcasing their interplay and integration. Through this interdisciplinary approach, we can pave the way for significant advancements in the field of visual texture analysis and its practical implications.

In addition to the theoretical contributions of this research, the code implementation will be made available for further exploration and replication. The code associated with this study will be published in a public GitHub repository, ensuring accessibility for researchers and practitioners interested in examining the methodologies employed.

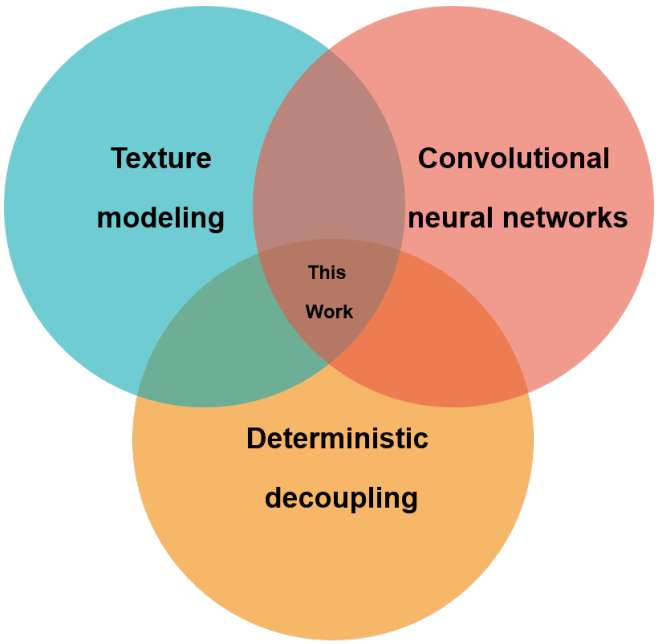

Figure 1.1: Venn diagram representing the main components of this work: Deterministic Decoupling, CNNs, and Texture Modelling. The diagram visually illustrates the interplay and integration of these three key elements.

1.2 Objectives

The main objective of this research is to explore the feasibility of integrating the powerful methodology of nested normalizations to decouple features, as proposed by Martínez-Enríquez, González, and Portilla [7], with Simoncelli's state-of-the-art texture modelling technique [6]. By combining these two approaches, we aim to enhance the descriptive power of features for visual texture analysis and representation.

To achieve this objective, we will develop a model that extracts spatial means from the channels of a pre-trained CNN. Subsequently, we will investigate the application of nested normalizations by layers to decouple these features. By integrating these two approaches, we strive to enhance the descriptive power of features for visual texture analysis and representation.

The secondary objectives of the research are as follows:

- **Review of Concepts**: Conduct a comprehensive review of the key mathematical, theoretical concepts and previous work related to decoupling features [8, 9, 7]. This step ensures a thorough understanding of the underlying principles behind the proposed techniques.
- **Feature Extraction**: Use a convolutional neural network architecture to extract meaningful features from a given texture. This process involves computing the spatial means of these features, which helps capture important characteristics of the texture. It is worth noting that the selected features do not meet the theoretical requirements for perfect decoupling. However, previous research [7] has demonstrated instances where despite not meeting these conditions, approximate results have still improved the descriptive power of the features.
- **Comparison of Feature Means**: Develop an effective method for comparing the spatial means of the extracted features. This comparison will generate a cost value, which we can differentiate with PyTorch, facilitating adjustments to the texture.
- **Trajectory Analysis of Gradients**: Develop methods to analyse and extract the trajectories of gradients from the feature maps of our pre-trained CNN. Investigate different techniques for capturing the temporal evolution and spatial variations of gradients during the style transfer process. This objective focuses on exploring how gradient trajectories can enhance the representation of style and texture information in images.
- **Performance Evaluation and Comparison**: Evaluate and compare the performance of the proposed approach with other existing methods for style transfer and texture synthesis. This evaluation offers valuable information about the advantages and disadvantages of the suggested method in comparison to alternative approaches.
- **Results Analysis and Discussion**: Analyse and discuss the results of the experiments, emphasizing the strengths and weaknesses of the proposed approach. Additionally, identify possible areas for improvement and future research directions.

By achieving these objectives, this research aims to contribute to the advancement of image style transfer and texture synthesis techniques and provide practical im-

plementations and insights into their effectiveness. Figure 1.2 represent the main factors taken into account to design our texture model, and Figure 1.3 illustrates how these statistical representations of textures can be used.

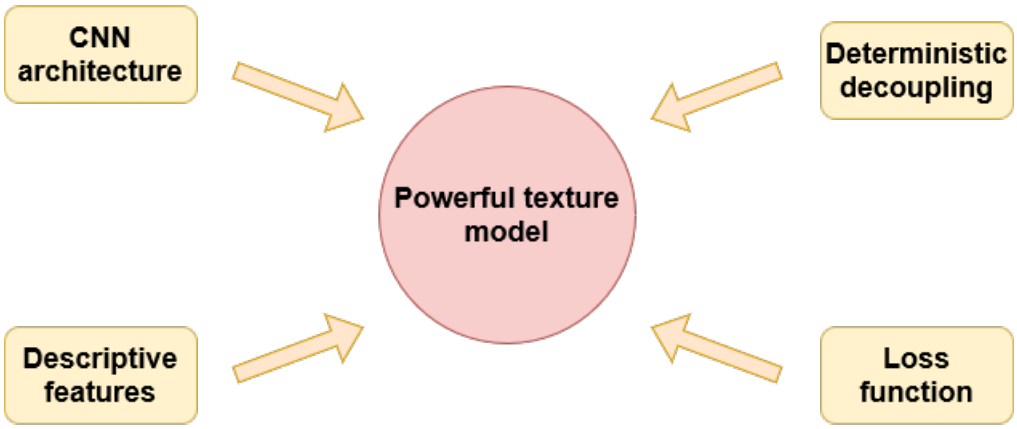

Figure 1.2: Conceptual diagram outlining the key factors considered for optimizing the performance of the texture representation model. First the utilization of a CNN architecture renowned for its superior image interpretation capabilities. Additionally, careful attention is given to the selection of features that effectively represent the underlying textures. To refine the training process, a robust loss function is designed, enabling precise and accurate optimization of the model's performance. Finally, the possible implementation of deterministic decoupling techniques to further enhance the descriptive power of the extracted features.

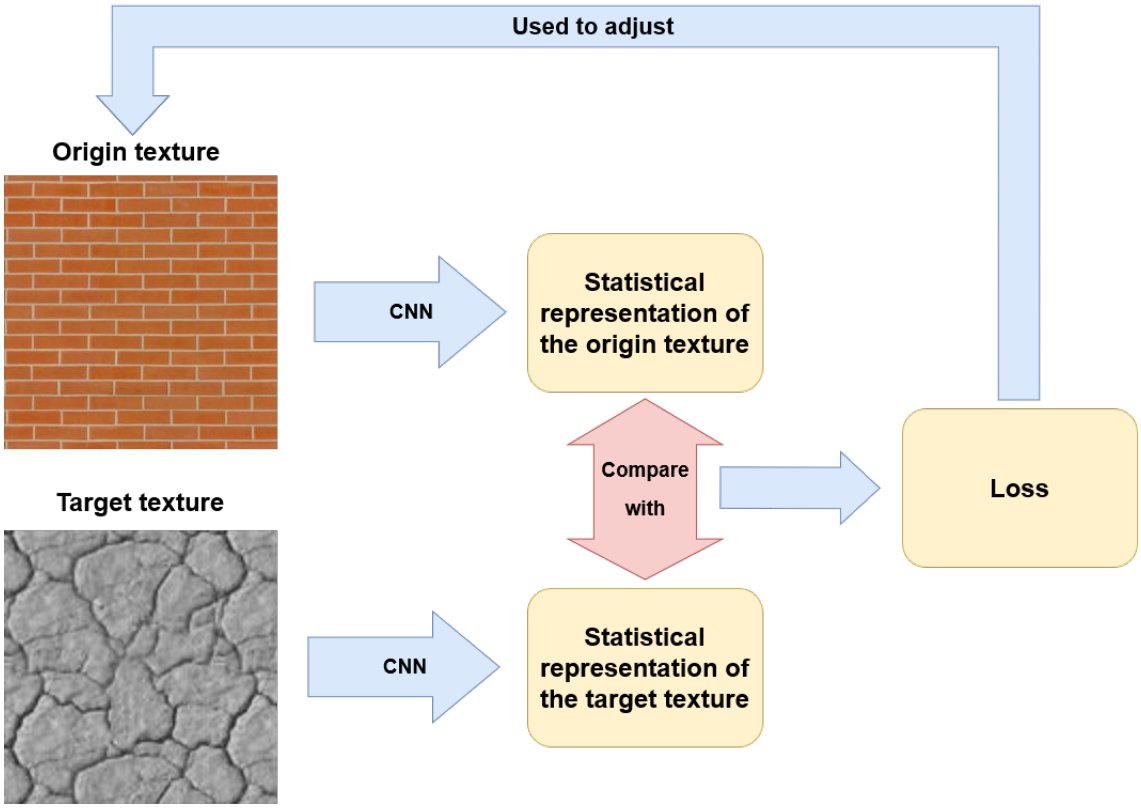

Figure 1.3: Illustration of the process utilizing statistical representations of textures for style transfer. By leveraging texture features, the original texture is transformed to align with the target texture, enabling compelling visual transformations. This emphasizes the importance of accurate statistical representations of textures, which motivates our exploration of deterministic decoupling.

Chapter 2

Theoretical Context

2.1 Feature Decoupling: Enhancing Descriptive Power

A feature refers to a measurable or observable property or characteristic of a data point or object. It represents a specific aspect or attribute that can be mathematically quantified or described. Features are typically represented as variables or dimensions that capture relevant information about the data, enabling analysis within the dataset.

For instance, in image analysis, features could include the presence of specific shapes or patterns, or even the distribution of texture within the image. These features capture measurable properties of the image and are represented as numerical values or descriptors. By quantifying these features, it is possible to train a machine learning model to learn patterns and make predictions based on the extracted features.

Feature decoupling refers to the process or technique of disentangling or separating the correlations among different features or attributes of a given data representation [9]. This allows us to understand each feature independently, control them individually, and expand their application possibilities.

The deterministic decoupling of features relies on their gradient vectors being orthogonal. If the gradient vectors of two features, denoted as f_i and f_j, are orthogonal within the domain Ω, it indicates that these features are mutually decoupled. These features are functions that operate on a vector x, which can represent various forms of data, including textures. By decoupling these features, we ensure that modifying one feature doesn't affect the others, as they capture distinct and unrelated information.

The objective of feature decoupling is to transform a set of coupled features into a new set where the gradients are mutually orthogonal. This transformation decreases redundancy and enhances descriptive power of the transferred features. Given an initial set of coupled features, denoted as $S = \{f_i : \Omega \to \mathbb{R}, i = 1...M\}$, our aim is to find a new set of features $\hat{S} = \{\hat{f}_i : \Omega \to \mathbb{R}, i = 1...M\}$, being identical to the original in a high dimensional manifold, $\mathbb{R}^1$, while simultaneously demonstrating orthogonal gradients in the whole domain, $\mathbb{R}^D$ [7].

To illustrate the impact of decoupling on feature representation, Figure 2.1 provides a visual comparison. On the left side, we observe regular coupled features, which

exhibit significant compression in the values of skewness and kurtosis. This compression makes it challenging to adjust one without modifying the other. In contrast, the decoupled features on the right side demonstrate more balanced and distinct variations in skewness and kurtosis values, allowing for more flexible and precise adjustments.

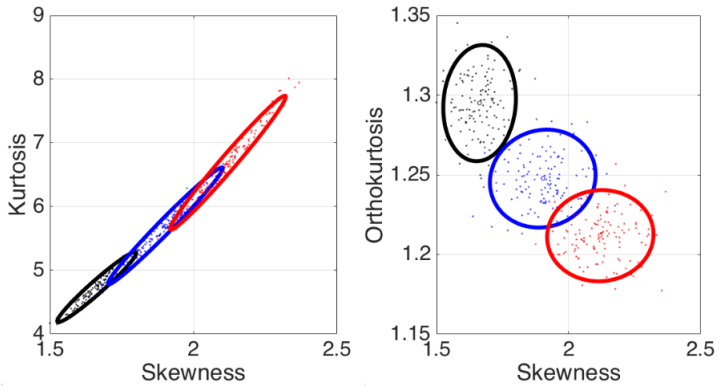

Figure 2.1: Comparison between regular coupled features (left) and decoupled features (right). The original coupled features exhibit significant overlap in the values of skewness and kurtosis, making it challenging to adjust one without modifying the other. Figure reproduced from [8].

By successfully achieving decoupling, we enable the independent variation and analysis of individual features within the given domain. This not only improves the efficiency of feature transfer processes, but also enhances the descriptive potential of the transferred features. Decoupled features encompass distinct information, enabling more informative and meaningful representations of the underlying data. This approach has been successful in improving performance for regression and classification problems [7].

It is worth noting that noteworthy results have been achieved even when the theoretical requirements for exact decoupling were not met. Specifically, the statistical moments of degree larger than two at the output of a set of filters.

This observation motivated us to explore the potential of the nested normalization decoupling technique within the context of CNNs, despite the fact that the gradient of the features does not satisfy the Frobenius integrability condition, which means it does not meet the theoretical requirements for perfect decoupling. Although perfect decoupling may be unattainable, approximate decoupling could still yield valuable outcomes. We were intrigued to investigate whether this method could provide valuable insights.

By iteratively applying the nested normalization decoupling method to the selected features across different layers of the CNN, our goal is to uncover meaningful relationships and patterns. Our particular focus is to assess whether this approach could enhance interpretability and potentially improve the performance of the models.

2.1.1 Normalization-based Decoupling Method

The nested normalization decoupling method, described in this section, is based on the work of Portilla, González, and Martínez [7]. This technique aims to facilitate the identification of a set of features that possess the desired properties outlined in Section 2.1 by using normalization as a decoupling mechanism.

To grasp the decoupling method, let's first delve into the concept of a gradient system. Consider a feature f defined on a connected open set Ω. The system's paths are represented by the trajectories of an initial value problem, which can be described by the following ordinary differential equation:

$$\begin{aligned} \frac{dx}{dt} &= -\nabla f(x), \\ x(0) &= x_0. \end{aligned} \tag{2.1}$$

The trajectory passing through x_0 is denoted as $I(x_0, f)$.

The objective of the decoupling method is to identify a transformation $\hat{x}(x) : \mathbb{R}^N \to \mathbb{R}^N$ that, in the case of having two features, it decouples feature $g(x)$ from $f(x)$. Specifically, the transformed feature $\hat{g}(x) = g(\hat{x}(x))$ is designed to be decoupled from $f(x)$, ensuring that the gradients of $f(x)$ and $\hat{g}(x)$ are orthogonal, i.e., $\nabla f(x) \cdot \nabla \hat{g}(x) = 0$.

The decoupling process involves selecting a reference value, denoted as v_{ref}, for the feature $f(x)$. The choice of v_{ref} can be tailored to the specific analysis requirements. For instance, setting v_{ref} as the mean of $f(x)$, such as $v_{\text{ref}} = 0$, is a common approach. The set of points satisfying $f(x) = v_{\text{ref}}$ is referred as the *reference manifold* $R(v_{\text{ref}})$.

Next, locally defined trajectories known as *invariant manifolds* are examined, denoted as $I(x)$. These trajectories, illustrated in Figure 2.2, are determined by the gradient system associated with the feature $f(x)$ and play a crucial role in the decoupling process.

The main objective is to find a transformation function $\hat{x}(x)$ that remains invariant under local gradient perturbations. This means that if two points x_1 and x_2 belong to the same invariant manifold $I(x)$, their transformed images should be identical, i.e., $\hat{x}(x_1) = \hat{x}(x_2)$.

To achieve this, the normalization, denoted as $\hat{x}_S(x; v_{\text{ref}})$, is defined as the function that maps a point x to the intersection point $\vec{x}_{\text{ref}}$ between its invariant manifold $I(x)$ and the reference manifold $R(v_{\text{ref}})$. This intersection is unique.

To simplify, constructing curves that follow the gradient involves moving along trajectories by following the local gradient of the feature f until the reference value v_{ref} is attained.

Given another feature g, we define the transformation $\hat{g}$ as:

$$\hat{g}(x) = g(\hat{x}(x)) \tag{2.2}$$

The normalization-based decoupling method provides a powerful framework for finding a set of decoupled features by using the concept of normalization and invariant manifolds. By following the proposed approach, we can effectively decouple features and facilitate further analysis and processing of the data.

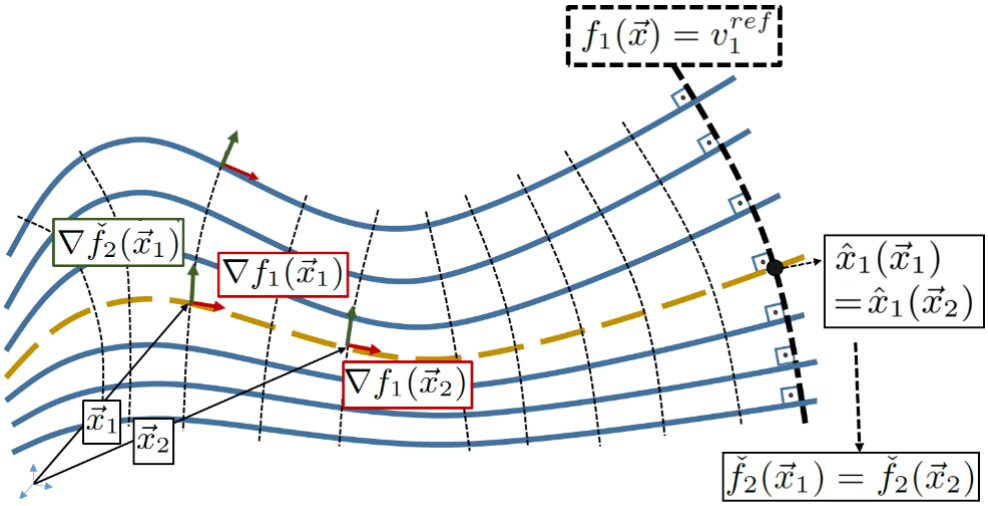

Figure 2.2: Decoupling features entails finding the intersection between the invariant manifold represented by the blue lines and the reference manifold represented by the dotted back line. Figure reproduced from [8].

To extend the decoupling process, the number of dimensions has to be expanded. This method follows the normalization scheme previously explained for two features, using the gradients of all features f_i. However, when constructing varieties based on multiple feature gradients, additional conditions outlined in Portilla's work must be fulfilled.

The concept of a mapping being invariant with respect to a set of features $S = \{f_i : \Omega \subset \mathbb{R}^N \to \mathbb{R}, i = 1, \ldots, M\}$ is introduced. A non-constant, differentiable mapping $y_S : \Omega \to \Omega$ is said to be invariant regarding S if and only if Equation 2.3 holds true:

$$J_{y_S}(x)\nabla f_i(x) = 0, \quad \forall f_i \in S, \forall x \in \Omega, \tag{2.3}$$

where J_{y_S} represents the Jacobian matrix of y_S.

Extending the dimensions involves the creation of multidimensional subvarieties using tangent hyperplanes determined by the gradients of the features. For every point in the N-dimensional space, denoted as $x_0 \in \mathbb{R}^N$, there exists a corresponding invariance variety defined by the tangent plane $I(x_0; S)$. This also applies to the reference value, which corresponds to an invariance variety called the reference variety $R_S(v_{\text{ref}})$.

An algorithm that obtains decoupled features in an iterative and straightforward manner is illustrated in Figure 2.3. The figure showcases concentric circles representing reference varieties, denoted as R_k ($k = 1, ..., M$). Each reference variety corresponds to a set of images with desired reference values for the original features from 1 to k. The nested structure of the circles, where R_{k+1} is a subset of R_k, signifies a progressive normalization process. During each iteration or layer, the images are normalized while maintaining the fixed reference values established in previous iterations.

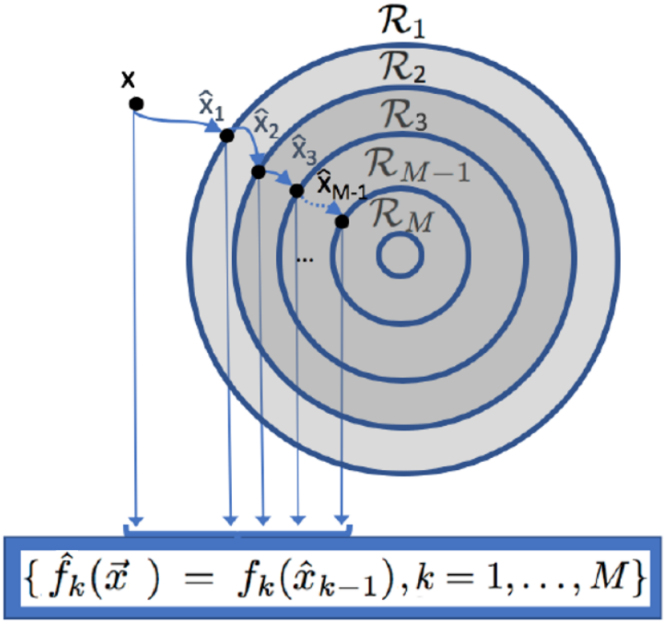

Figure 2.3: Algorithmic Representation of Decoupling for Multiple Features. This figure illustrates the step-by-step process of orthogonally decomposing multiple features. The algorithm aims to decouple newly acquired features from the previously decoupled ones, iteratively progressing towards a final set of features with mutually orthogonal gradients. Each iteration represents a distinct stage in the decomposition process, contributing to the overall achievement of a collection of decoupled features. Reproduced from [7].

The new features are defined as the original features measured in the normalized samples. With each normalization step applied to an original feature, a new decoupled feature is obtained while retaining the previously normalized features.

The normalization of a set of features is achieved through the intersection of the invariance variety and the reference variety. Specifically, for a given point x and a reference value v_{ref}, the normalized feature set $\hat{x}_S(x; v_{\text{ref}})$ is defined as:

$$\hat{x}_S(x; v_{\text{ref}}) = I(x; S) \cap R(v_{\text{ref}}) \tag{2.4}$$

According to the demonstrations by Portilla, González, and Martínez, if a trajectory $I(x, S)$ exists, then any differentiable function $g(x)$ satisfies $g(\tilde{x}S(x))$ being decoupled from S. This property validates the effectiveness of the proposed approach.

The authors demonstrate that by iteratively applying this normalization process, we can obtain progressively decoupled features.

2.2 Using a Modified DISTS Model for Describing Textures

In the previous sections, we discussed the significance of decoupling features to enhance their descriptive power. However, it is essential to start with features that already exhibit strong descriptive capabilities, as this forms a solid foundation for improvement. This raises the question: How can we obtain features that effectively capture the underlying texture? To tackle this challenge, we turn to convolutional neural networks, particularly the renowned DISTS model developed by Simoncelli et al. [6], which represents a cutting-edge approach in this domain.

CNNs are deep learning models specifically designed for image analysis, capable of automatically extracting meaningful patterns and characteristics from visual data. In our case, our objective is to extract features that accurately represent image textures.

The DISTS model utilizes a pre-trained VGG-16 CNN with frozen weights for feature extraction, with one notable modification: the replacement of max pooling operations with weighted L_2 pooling. Weighted L_2 pooling is a specific pooling operation that addresses the issue of aliasing artefacts introduced by max pooling in certain scenarios [10]. By combining elements from the pooling neighbourhood using weighted averaging, Weighted L_2 pooling effectively blurs the image and reduces high-frequency components before subsampling, thus preventing aliasing artefacts.

The feature representation of image texture in the DISTS model is based on the spatial means of feature maps obtained from the pre-trained VGG-16 CNN. In a CNN, multiple layers are stacked, each containing several channels. A feature map refers to the output of a particular convolutional layer. The spacial means of these feature maps serves as the primary representation of texture. Prior to feeding the image into the network, it undergoes normalization by subtracting the mean and dividing by the standard deviation of a texture database. The network consists of five stages, and the output from each stage is utilized.

To ensure an injective mapping in the modified VGG architecture, the input image is included as an additional feature map, specifically as the "zeroth" layer of the

network. Injectivity guarantees that distinct inputs result in distinct outputs, which is an essential property for the proper behaviour of the final representation.

For our project, we introduce two main modifications to the DISTS model. Firstly, we incorporate the correlation term between colour channels as a feature map. This consideration allows the model to account for the level of association or relationship between colours, effectively preventing the presence of high-frequency patterns or artefacts in the textures. The representation includes the input image, correlations, and channel outputs at each of the five stages, resulting in a concatenated list of seven elements. Special averages are calculated for each element in the representation.

While the original evaluation framework considers both structure and texture information to assess image quality, our specific project focuses solely on texture representation. Consequently, the second modification tailors the model for the assessment of textures, disregarding structure information.

We will use the following notation. The CNN consists of I layers, indexed by i, with each layer containing $J(i)$ channels, indexed by j. The feature representation is defined as $\mu_{(i,j)}(x), i = 1 \dots I, j = 1 \dots J(i)$, which represents the spatial means of channel j for layer i. These values are computed for a target texture, denoted as Y target, resulting in a set of values $v^{des}_{(i,j)} = \mu_{(i,j)}(y_{tgt})$. This feature representation allows for a comprehensive characterization of image textures, enabling various applications in image analysis and processing.

In summary, the modified DISTS model offers a powerful approach for describing textures by utilizing a pre-trained VGG-16 CNN with weighted L_2 pooling. The feature representation based on spatial means of the feature maps, including the incorporation of the correlation term between colour channels, provides a robust representation of image textures. This feature representation captures intricate details and associations within textures, enabling accurate and meaningful characterization.

2.2.1 Loss Function for Texture Features

To quantify the dissimilarity between texture features, we apply a new loss function to the modified DISTS model. This loss function enables the measurement of the discrepancy between the origin texture features of the examined images and the target features.

The cost function, denoted as $C(x)$, measures the difference between the input image x and the desired target feature values. It is formulated as the sum of squared differences between the means of each channel in each layer. To ensure that the cost function is appropriately scaled, normalization is applied. The sum is divided by the total number of layers I and further divided by the number of channels in each layer $J(i)$.

$$C(x) = \frac{1}{I} \sum_{\forall i \text{ capa}} \frac{1}{J(i)} \sum_{\forall j \text{ canal} \in \text{ i capa}} \frac{\left(\mu_{(i,j)}(x) - v^{des}_{(i,j)}\right)^2}{c_{(i,j)} + (v^{des}_{(i,j)})^2} \tag{2.5}$$

In Equation 2.5, the cost function $C(x)$ is calculated by considering the difference between the origin features and the target features. To account for the magnitude of

the desired output, the squared target features are used as a normalization factor.

The term $c_{(i,j)}$ represents small constants introduced to prevent singularities in the cost function when the mean of a channel is zero everywhere. These constants act as regularization terms, ensuring stability and avoiding numerical issues during the calculations. By adding these constants, potential problems arising from division by zero are mitigated, allowing for smooth and reliable computations.

Our version of the DISTS model uses the VGG16 CNN, which employs the rectified linear unit (ReLU) activation function. With ReLU non-linearities, a channel's mean is zero only if the entire channel consists of zeros.

To refine the cost function, we normalized by dividing by the number of layers and channels. By normalizing the cost function, we promote balanced convergence in terms of relative error for each channel within each layer and across different layers. This normalization step facilitates effective training and enhances the overall performance of the model.

The deliberate design of the cost function ensures that its gradient becomes a linear combination of the feature gradients. This property is achieved by intentionally selecting a quadratic dependency in the cost function. Taking partial derivatives of Equation 2.5 further illustrates this property:

$$\begin{aligned} \nabla C(x) &= \sum_{i=1}^{I} \sum_{j=1}^{J(i)} \alpha_{(i,j)}(x) \nabla \mu_{(i,j)}(x) \\ \alpha_{(i,j)}(x) &= \frac{2}{I \cdot J(I)} \frac{\mu_{(i,j)}(x) - v_{(i,j)}^{des}}{c_{(i,j)} + (v_{(i,j)}^{des})^2} \end{aligned} \tag{2.6}$$

As a result, the process of minimizing the cost through gradient descent can be visualized as following a trajectory within the invariant submanifold, which is shaped by the local linear combinations of the gradients. A concept integral to the nested normalization process.

2.3 Runge-Kutta Method for Solving Gradient Systems

When dealing with gradient systems obtained from cost functions, it is often challenging to find analytical solutions for the resulting ordinary differential equations (ODEs). As a result, alternative methods are required to solve such problems. Gradient descent methods, such as Adam or stochastic gradient descent, are commonly employed for this purpose. These methods iteratively update the model's parameters by following the direction of the negative gradient, allowing convergence towards optimal solutions.

However, in our specific case, closely tracking the trajectory defined by the gradient is crucial for potentially obtaining decoupled features. Therefore, we explore the use of a numerical method to approximate the solutions of the ODEs. One widely employed numerical integration technique is the Runge-Kutta method, which offers accurate numerical approximations [11] which we will use to ensure stability and adherence

to the invariant submanifold. Our experiments will incorporate either Adam or the Runge-Kutta method, depending on the specific situation.

The ODE to be solved, referred to as a gradient system, is given by:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \tag{2.7}$$

Here, y represents an unknown function of time t that needs to be approximated. The rate of change of y, $\frac{dy}{dt}$, depends on both t and y itself, as governed by the function f. The initial conditions, denoted as t_0 and y_0, define the starting point for the solution.

To perform numerical integration, we select a step size $h \geq 0$ and use the following iterative formulas:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\ k_4 &= f\left(t_n + h, y_n + hk_3\right). \end{aligned} \tag{2.8}$$

Here, t_n represents the current time, y_n represents the current approximation of y, and k_1, k_2, k_3, and k_4 represent intermediate values based on the slopes at different points within the interval.

The approximation is then updated using the formula:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.9}$$

This equation corresponds to the Runge-Kutta fourth order (Runge-Kutta 4) method, providing an approximation, y_{n+1}, for the solution $y(t_{n+1})$. At each step, the Runge-Kutta 4 method computes the next value y_{n+1} based on the present value y_n and a weighted average of four increments. Each increment is determined by the estimated slope at specific points within the interval.

In addition to the aforementioned method, we incorporate the Dormand-Prince method of the Runge-Kutta family, which dynamically optimizes the step size h without requiring manual intervention. This parameter adaptation eliminates the need for a fixed value to be set at the beginning of the computation. The code implementation of the Dormand-Prince method using PyTorch can be obtained from the GitHub repository maintained by Ricky T. Q. Chen [12]. However, upon evaluation, the results obtained using the Dormand-Prince method were found to be either identical or even inferior to those achieved using the standard Runge-Kutta 4 method. Furthermore, it was noted that the Dormand-Prince approach is significantly more computationally demanding. Consequently, all subsequent experiments involving Runge-Kutta computations in this thesis will exclusively employ the Runge-Kutta 4 version.

The Runge-Kutta method provides an effective approach for obtaining accurate and stable numerical approximations of ODE solutions. It allows for efficient solution of gradient systems, yielding results that closely resemble the ideal solution while remaining within the invariant submanifold.

Chapter 3

Computational Context

In line with the goal of promoting reproducibility and further research, the code for the implementation discussed in this chapter will be made publicly available. The complete codebase, along with documentation, can be accessed at the following GitHub link: `https://github.com/DanielSalvado/CNN-deterministic-decoupling`. Access to the code repository will facilitate a deeper understanding of the methodologies employed and enable others to replicate and build upon this research.

3.1 Model Implementation

In this section, we describe the implementation details of our model, `DISTS_mod`, which is implemented as a modification of the PyTorch version of DISTS [6]. We provide an overview of the code structure, explain the key components and logic, and highlight important modifications made to existing algorithms.

```
class DISTS_mod(torch.nn.Module):
    def __init__(self, load_weights=True):
        super(DISTS, self).__init__()

        vgg_pretrained_features = models.vgg16(pretrained=True).features
        self.stage1 = torch.nn.Sequential()
        self.stage2 = torch.nn.Sequential()
        self.stage3 = torch.nn.Sequential()
        self.stage4 = torch.nn.Sequential()
        self.stage5 = torch.nn.Sequential()

        for x in range(0,4):
            self.stage1.add_module(str(x), vgg_pretrained_features[x])

        self.stage2.add_module(str(4), L2pooling(channels=64))
        for x in range(5, 9):
            self.stage2.add_module(str(x), vgg_pretrained_features[x])

        self.stage3.add_module(str(9), L2pooling(channels=128))
        for x in range(10, 16):
            self.stage3.add_module(str(x), vgg_pretrained_features[x])

        self.stage4.add_module(str(16), L2pooling(channels=256))
        for x in range(17, 23):
            self.stage4.add_module(str(x), vgg_pretrained_features[x])

        self.stage5.add_module(str(23), L2pooling(channels=512))
        for x in range(24, 30):
            self.stage5.add_module(str(x), vgg_pretrained_features[x])
```

```
30
31          self.stages=[self.stage1, self.stage2, self.stage3, self.stage4, self.stage5]
32          for param in self.parameters():
33              param.requires_grad = False
34          self.register_buffer("mean", torch.tensor([0.485, 0.456, 0.406]).view(1,-1,1,1))
35          self.register_buffer("std", torch.tensor([0.229, 0.224, 0.225]).view(1,-1,1,1))
36
37          self.chns =  [3,3,64,128,256,512,512]
38          self.target=None
```

The model is a subclass of torch.nn.Module and is designed to extract feature maps from input images using the VGG16 model's pre-trained weights. It consists of five stages, named stage1 to stage5, which are sequential containers that hold the layers or modules of the model.

Each stage represents a specific portion of the VGG16 architecture and performs sequential computations on the input image. The layers from the VGG16 model are added to the respective stages using `add_module`. Additionally, max pooling is exchanged by weighted L_2 pooling.

After defining the stages, they are stored in a list called `self.stages` for easy access and further processing.

The DISTS model also registers the mean and standard deviation values from a database as buffers, which are used for normalizing the input images. These values ensure consistent normalization across different images.

Furthermore, the code sets the `requires_grad` attribute of all model parameters to False, which effectively freezes the parameters and prevents them from being updated during training. This is done to ensure that the pre-trained VGG16 weights remain unchanged. Instead of modifying the network weights to improve its response to a static image, our approach involves manipulating the image itself so that it elicits a better response from the fixed network.

The `chns` list represents the number of channels in each stage of the model.

Lastly, the target variable is initialized as None, as it will be set up later

3.1.1 Model functions

This section presents an overview of the key functions implemented in the `DISTS_mod` model. Each function plays a specific role in the model's functionality.

3.1.1.1 `set_target`

The `set_target` function establishes a target image with which the distance is to be calculated. It takes the target image as input and computes the mean of the feature maps for each channel in the target image. These mean values are stored in `feats1Mean` and are used for subsequent calculations.

```
1
2 def set_target(self,target):
3       '''establishes a target image with which the distance is to be calculated'''
4       feats1 = self.forward_once(target)
5       feats1Mean=[]
6       for k in range(len(self.chns)):
7         feats1Mean.append(feats1[k].mean([2,3], keepdim=True))
```

```
8
9       self.r_means=feats1Mean
10      self.target=target
```

3.1.1.2 `forward`

The `forward_once` function executes a part of the forward pass on the neural network for an input image `x` and returns the intermediate outputs representing the image in the model.

First, the input image is normalized, and a tensor `xfcov` is initialized to store the energy and correlation factors for each channel. Each channel of the normalized image is squared and assigned to the corresponding energy factor in `xfcov`. The cross-channel correlation factors are calculated and stored in `xfcov` as well.

The normalized image then goes through each stage of the model, and the intermediate outputs at different layers are stored. Finally, a list containing the original input `x`,`xfcov`, and the intermediate outputs from each stage is returned.

```
1     def forward_once(self, x):
2       ''' Perform a forward pass on the neural network for an input x
3        and returns the intermediate outputs of each network layer
4       '''
5       Ny = x.shape[2]
6       Nx = x.shape[3]
7
8       xfcov = torch.zeros([1, 6,Ny,Nx], dtype=x.dtype, device=x.device)
9       h = (x - self.mean) / self.std  # mean and std are typical values
10      xfcov[0, 0, :, :] = h[0, 0, :, :] * h[0, 0, :, :]  # energy R
11      xfcov[0, 1, :, :] = h[0, 1, :, :] * h[0, 1, :, :]  # energy G
12      xfcov[0, 2, :, :] = h[0, 2, :, :] * h[0, 2, :, :]  # energy B
13      xfcov[0, 3, :, :] = h[0, 0, :, :] * h[0, 1, :, :]  # correlation factor RG
14      xfcov[0, 4, :, :] = h[0, 0, :, :] * h[0, 2, :, :]  # correlation factor RB
15      xfcov[0, 5, :, :] = h[0, 1, :, :] * h[0, 2, :, :]  # correlation factor GB
16
17      # Pass through each stage of the model and store intermediate outputs
18      h = self.stage1(h)
19      h_relu1_2 = h
20      h = self.stage2(h)
21      h_relu2_2 = h
22      h = self.stage3(h)
23      h_relu3_3 = h
24      h = self.stage4(h)
25      h_relu4_3 = h
26      h = self.stage5(h)
27      h_relu5_3 = h
28
29      return [x,xfcov,h_relu1_2, h_relu2_2, h_relu3_3, h_relu4_3, h_relu5_3]
```

The `forward` function performs a forward pass on the neural network for an input image `x` and computes the distance between the input features and a pre-defined target. This function serves as a measure of similarity between the input features and the target, operating at multiple layers of the model.

During execution, the function calculates the mean of the input feature maps for each channel within each layer. It then applies the cost function, which quantifies the distance between the mean values. The resulting distance scores are accumulated across layers and divided by the total number of layers to obtain a comprehensive similarity score.

```
    def forward(self, x):
      '''Performs a forward pass on the neural network for an input x
        and return the distance between the input features and a preset target
      '''
      layers = len(self.chns)
      feats0 = self.forward_once(x)
      dist1 = 0
      c1 = 1e-6
      feats1Mean=self.r_means

      for k in range(layers):
          x_mean = feats0[k].mean([2,3], keepdim=True)
          y_mean = feats1Mean[k]
          S1 = (x_mean-y_mean)**2 / (c1 + y_mean**2)
          dist1 = dist1 + S1.sum(1,keepdim=True)/self.chns[k]

      dist1 = dist1/len(self.chns) # number of layers
      score = dist1.squeeze()
      return score
```

The similarity score obtained through the forward pass and the calculation of the distance provides a quantitative measure of the likeness between the input features and the pre-defined target. This score plays a crucial role in assessing the level of resemblance, enabling valuable insights into the similarity and dissimilarity patterns within the neural network's intermediate representations.

3.2 Runge-Kutta implementation

The following code extract presents the implementation of the RungeKutta4 function, which enables numerical integration of ODEs using the fourth-order Runge-Kutta method. The function accepts the current value of `x`, a user-defined ODE function `c`, the step size for numerical integration, and a list to store the resulting values.

Within the RungeKutta4 function, the fourth-order Runge-Kutta method is executed in a step-wise manner. It involves evaluating the ODE function c at four different points `x1, x2, x3, x4` and determining their corresponding gradients `k1, k2, k3, k4`. These gradients are computed by employing automatic differentiation to back-propagate through the ODE function. The function then applies the fourth-order Runge-Kutta formula to update the value of `x`. In this study, the ODE function `c` will correspond to the forward function responsible for calculating the loss, since that is the one we are trying to integrate.

```
def RungeKutta4(x, c, step_size,list):
    '''
    Runge-Kutta method of fourth order for solving differential equations.
    Calculates the value of the function at four different points and the corresponding
        gradients.
    Finally, updates the value of x using the fourth-order Runge-Kutta formula.
    '''

    x1 = x.detach().clone().requires_grad_(True)
    y1 = c(x1)
    list.append(y1.item())
    y1.backward()
    k1 = -1 * x1.grad.clone()
    x1.grad.data.zero_()

    x2 = (x1 + k1 * step_size / 2).detach().clone().requires_grad_(True)
    y2 = c(x2)
```

```
    y2.backward()
    k2 = -1 * x2.grad.clone()
    x2.grad.data.zero_()

    x3 = (x1 + k2 * step_size / 2).detach().clone().requires_grad_(True)
    y3 = c(x3)
    y3.backward()
    k3 = -1 * x3.grad.clone()
    x3.grad.data.zero_()

    x4 = (x1 + k3 * step_size).detach().clone().requires_grad_(True)
    y4 = c(x4)
    y4.backward()
    k4 = -1 * x4.grad.clone()
    x4.grad.data.zero_()

    with torch.no_grad():
        x += ((k1 + 2 * k2 + 2 * k3 + k4) * step_size / 6)
```

The code extract demonstrates the RungeKutta4 method, a powerful numerical integration technique for solving ODEs using the fourth-order Runge-Kutta method. While optimization methods like Stochastic Gradient Descent (SGD) or Adam excel at optimizing high-dimensional parameter spaces efficiently, RungeKutta4 helps integrate gradients while minimizing deviation from corresponding manifolds. This is crucial for potentially obtaining decoupled features.

Chapter 4

Experiments

4.1 Texture Synthesis

In order to evaluate the potential and effectiveness of our proposed methodology, we conduct an initial experiment focusing on texture synthesis. Texture synthesis experiments aim to generate visually compelling textures using advanced computer graphics techniques. In our specific approach, we sought to generate textures solely using a noise image and the feature means extracted from a target texture. This experiment holds particular significance as it serves as a litmus test for validating the viability of our approach.

By successfully recreating visually appealing textures solely based on the feature means of the target texture, we can ascertain the accuracy and efficacy of our chosen methodology. This achievement would demonstrate the capability of our convolutional neural network to capture and represent the essential descriptive characteristics of an image through the mean values of its channels. Moreover, it would affirm that our approach provides a potent and reliable representation that encapsulates the visual essence of the target texture.

This texture synthesis experiment serves as a crucial milestone in our research, as its outcome will have implications for the subsequent stages of our work. A successful result would instil confidence in the feasibility of our proposed methodology and set the stage for further exploration and refinement of our approach. Conversely, any challenges or limitations encountered during this experiment would inform us of necessary adjustments and prompt further investigation.

The texture synthesis experiment begins by selecting an initial texture as a target. This target texture goes through a CNN to extract its feature means, which capture the essential characteristics of the texture. These feature means are saved for later use.

Next, a random noise image is generated to serve as the starting point for the synthesis process. This noise image acts as the canvas onto which the texture will be synthesized. To begin synthesizing the texture, the noise image is passed through the same CNN used in the previous step to extract its features.

Our cost function is then employed to evaluate the dissimilarity between the extracted features of the noise image and the feature means of the target texture. This com-

parison yields an error value that quantifies the degree of mismatch between the two sets of features.

Using PyTorch's automatic gradient calculation capabilities, the error value obtained from the cost function is integrated to compute the gradient. This gradient provides information on how to adjust the noise image to minimize the error and bring it closer to the target texture. By applying the computed gradient, the noise image is updated, modifying its pixel values based on the gradient information.

The adjustment process is iteratively repeated, with the noise image being passed through the CNN to extract features, the cost function being evaluated, and the noise image being adjusted based on the computed gradient. At this stage our only objective is to minimize the cost function, for that reason Adam will be used as the optimizer. This process is illustrated in Figure 4.1. While the iterative refinement process technically has the potential to continue indefinitely, we choose to conclude the process after a substantial number of iterations. This decision is based on the observation that, over time, the reduction of the loss becomes minimal or significantly slows down.

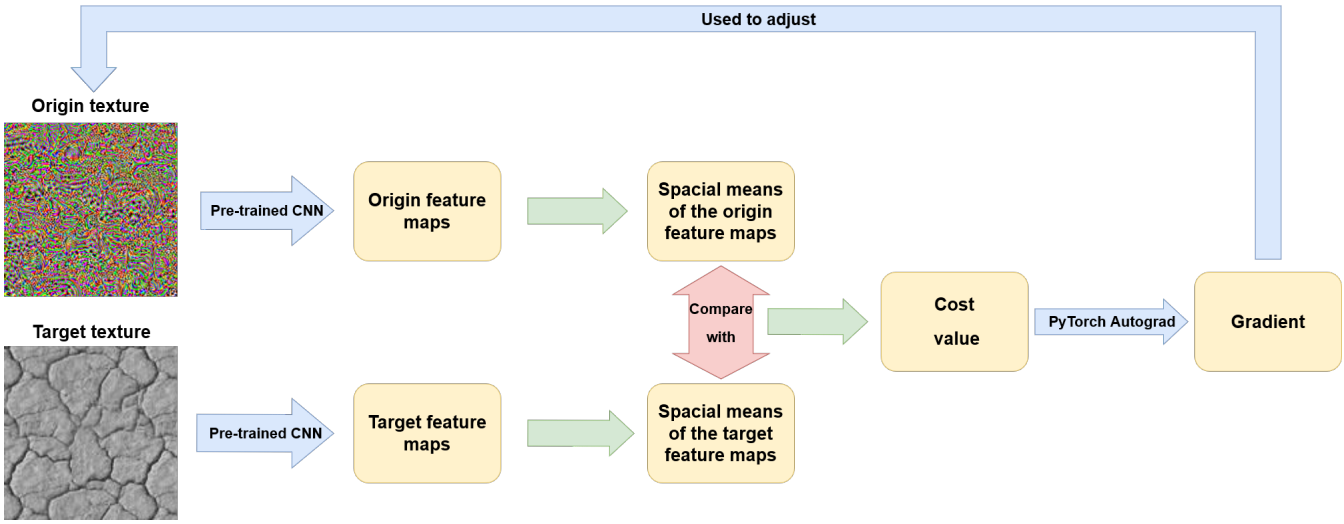

Figure 4.1: The diagram illustrates the process of texture synthesis we aim to implement, where we leverage the feature means of textures to transform the origin texture and align it with the appearance of the target texture. By substituting the origin texture with a real image, this concept can be extended beyond texture synthesis to address other challenges like style transfer. This versatile approach unlocks a multitude of possibilities for creating visually stunning and diverse outputs.

The code snippet below provides an implementation of the texture synthesis algorithm using the `DISTS_mod` model and the Adam optimizer:

```
1
2 #Prepare the target texture and initial noise image
3 target = prepare_image(Image.open(texture4.jpg").convert("RGB"),size=256)
4 origin=prepare_image(Image.open(noise.jpg").convert("RGB"),size=256)
5
6 #Instantiate the DISTS_mod model and set the target
7 model_d = DISTS_mod()
8 model_d.set_target(target)
9 epoch=0
```

```
initial_Loss=model_d.forward(origin).item()
LossList=[initial_Loss]

#Define the optimizer and stopping criterion
optimizer = torch.optim.Adam([{"params": [origin],"lr": 0.001}])
max_epochs=150000

#Iteratively adjust the noise image
while epoch < max_epochs:
    optimizer.zero_grad()
    loss=model_d.forward(origin)
    loss.backward()
    optimizer.step()
    LossList.append(loss.item())
    epoch+=1
```

4.2 Style Transfer

In the upcoming style transfer experiment, our objective is to investigate the manipulation of textures using the features derived from another source. This process bears resemblance to the texture synthesis experiment discussed earlier. However, instead of using a random noise image as the starting point, we will employ a real texture image. This modification introduces a fresh perspective into the style transfer process, enabling us to witness the interplay between different textures and artistic styles.

To initiate the style transfer experiment, we begin by selecting a target image that represents the artistic style we wish to incorporate into the final synthesized image. The target image is passed through a CNN to extract its feature means, which encapsulate the stylistic characteristics.

Next, we choose a different texture. This origin image is also passed through the same CNN to extract its feature means, capturing its information.

We then proceed to evaluate a cost function that measures the dissimilarity between the feature means of both images. This cost function calculates an error value that quantifies the discrepancy between the features.

Using PyTorch's automatic gradient calculation capabilities, the error value obtained from the cost function is integrated to compute the gradient. This gradient guides the adjustment of the origin texture image, allowing us to iteratively modify that texture to minimize the error and bring it closer to the desired style.

The adjustment process is repeated iteratively, with the origin texture image being passed through the CNN to extract its feature means, the cost function being evaluated, and the texture image being adjusted based on the computed gradient. Given the similarity to texture synthesis, we adopt a comparable stopping criterion for the style transfer experiment. Initially, we set a substantial number of iterations, which will be further adjusted based on the evolution of the optimization process. Additionally, we continue to utilize the Adam optimizer.

Through this style transfer experiment, we can witness the artistic interplay between different textures and content. The integration of the target image's features into the origin texture image fosters the creation of a synthesized image that encapsulates some of the stylistic essence of both.

Below is an example implementation of the style transfer:

```

# Load textures and convert them to RGB format
target = prepare_image(Image.open(texture4.jpg").convert("RGB"),size=256)
origin=prepare_image(Image.open(texture5.jpg").convert("RGB"),size=256)

#Instantiate the DISTS_mod model and initial values
model_d = DISTS_mod()
model_d.set_target(target)
epoch=0
initial_Loss=model_d.forward(origin).item()
LossList=[initial_Loss]

# Define the optimizer and stopping criterion
optimizer = torch.optim.Adam([{"params": [origin],"lr": 0.001}])
max_epochs=308000

#Iteratively adjust the texture style
while epoch < max_epochs:
    optimizer.zero_grad()
    loss=model_d.forward(origin)
    loss.backward()
    optimizer.step()
    LossList.append(loss.item())
    epoch+=1
```

4.3 Reversibility

The purpose of this experiment is to explore the feasibility of traversing trajectories defined by the gradient of our cost function and demonstrate reversibility through texture manipulation. The objective is to move both forward and backward along the same path by successfully integrating the gradient and following the defined trajectory. If we are able to move both forward and backward along the same path, this will mean that moving across invariant manifolds is possible, and thus we can decouple the features. Full reversibility, is a necessary condition to guarantee the existence of the submanifold of invariance and, therefore, of a well-defined normalization. This reversibility in adjusting textures will be achieved using the following methodology.

To closely follow the defined trajectory both forward and backward, the Runge-Kutta method will be utilized for numerical solution of the gradient. Additionally, an experiment will be performed using the Adam optimization algorithm, which aims to minimize the cost function quickly without explicitly considering accurate trajectory following. By comparing the results obtained from the Runge-Kutta method and Adam, the extent of reversibility offered by Runge-Kutta can be determined. The success of the experiment will be evaluated based on this comparison.

The experiment consists of two main steps:

1. **Style Transfer using Loss Function Adjustment**: Style transfer is performed on a given texture using the cost function as a loss value and adjusting according to the gradient of that loss. This process enables the transfer of the style from an origin texture to the target texture, up to a certain degree. The following code demonstrates the implementation of this step:

```


# Load the textures and set target
target=prepare_image(Image.open(path+"/imagenes/texture4.jpg").convert("RGB"),size=128)
```

```
origin=prepare_image(Image.open(path+"/imagenes/texture5.jpg").convert("RGB"),size=128)
x0=origin.detach().clone()
model_d.set_target(ref)

# Set the initial values
epoch=0
step_size = 0.1
turning_point=0.1

ErrorList=[math.sqrt(((x0-origin)**2).mean())/math.sqrt((x0**2).mean())]
initial_Loss=model_d.forward(origin).item()
LossList=[initial_Loss]
epoch+=1


# Run style transfer iterations
while LossList[-1] > turning_point:
    RungeKutta4(origin,model_d.forward,step_size,LossList)
    error = math.sqrt(((x0-origin)**2).mean())/math.sqrt((x0**2).mean())
    ErrorList.append(error)
    epoch+=1
print("Turning point reached")
```

2. **Texture Exchange and Continued Adjustment**: After achieving the desired level of style transfer, we proceed to exchange the target texture with the original texture. At this stage, our focus shifts to adjusting the image based on its original features, thus aiming to restore it as closely as possible to its original state. The adjustment of the texture is continued using the same methodology as before. The convergence behaviour is assessed by monitoring the proximity of the adjusted texture to the original texture. The reversion process is stopped when further iterations no longer bring the adjusted texture closer to the original texture, ensuring that the adjusted texture no longer approached the original texture. Here is the code that directly implements this step:

```

    # Perform texture exchange and continued adjustment
    model_d.set_target(x0)
    last_error=ErrorList[-1]

    # Run style transfer iterations in the opposite direction
    while ErrorList[-1] >= last_error:
        last_error=ErrorList[-1]
        RungeKutta4(origin,model_d.forward, step_size,LossList)
        error = math.sqrt(((x0-origin)**2).mean())/math.sqrt((x0**2).mean())
        ErrorList.append(error)
        epoch+=1
    print(END)
```

This experiment validates the reversibility of the adjustment process and its correspondence to traversing a differential submanifold. Additionally, it provides an evaluation of the performance of style transfer using the proposed gradient-based method. Analysing the results provides insights into the behaviour and properties of gradient-based optimization in texture-related tasks and evaluates the effectiveness of this approach for style transfer.

Chapter 5

Results and Conclusions

5.1 Texture Synthesis

In this section, we present the results of the texture synthesis experiment and draw conclusions based on the outcomes. The experiment aimed to generate visually compelling textures solely based on a noise image and the feature means extracted from a target texture. The positive results obtained from this experiment are crucial for the subsequent stages of our research.

The texture synthesis experiment yielded highly positive results, demonstrating the effectiveness of our proposed methodology. Through the iterative refinement process, we successfully synthesized textures that closely resemble the target texture, as evidenced by significantly minimized error values. Figure 5.1 showcases three synthesized textures alongside their corresponding target textures.

The synthesized textures in Figure 5.1 exhibit a high degree of visual resemblance to their respective target textures, effectively capturing intricate details, colour palettes, and overall texture patterns. These results validate the efficacy of our convolutional neural network to represent the essential characteristics of textures through the mean values of their channels.

Figure 5.2 illustrates the noise image utilized in our experiment. We specifically designed this noise image to stimulate all channels, as we observed that channels close to zero tend to remain "dead" and do not undergo significant changes during the adjustment process. By ensuring that our synthesis process had access to the full expressive capacity of the neural network and the target texture, we were able to generate visually compelling textures.

The positive outcome of this experiment not only confirms the efficacy of our approach, but also provides valuable insights into its capabilities and potential. With the successful texture synthesis as a foundation, we can extend our methodology to address various challenges, such as style transfer, by leveraging the features extracted from different textures. This versatility opens up a multitude of possibilities for creating visually captivating and diverse outputs.

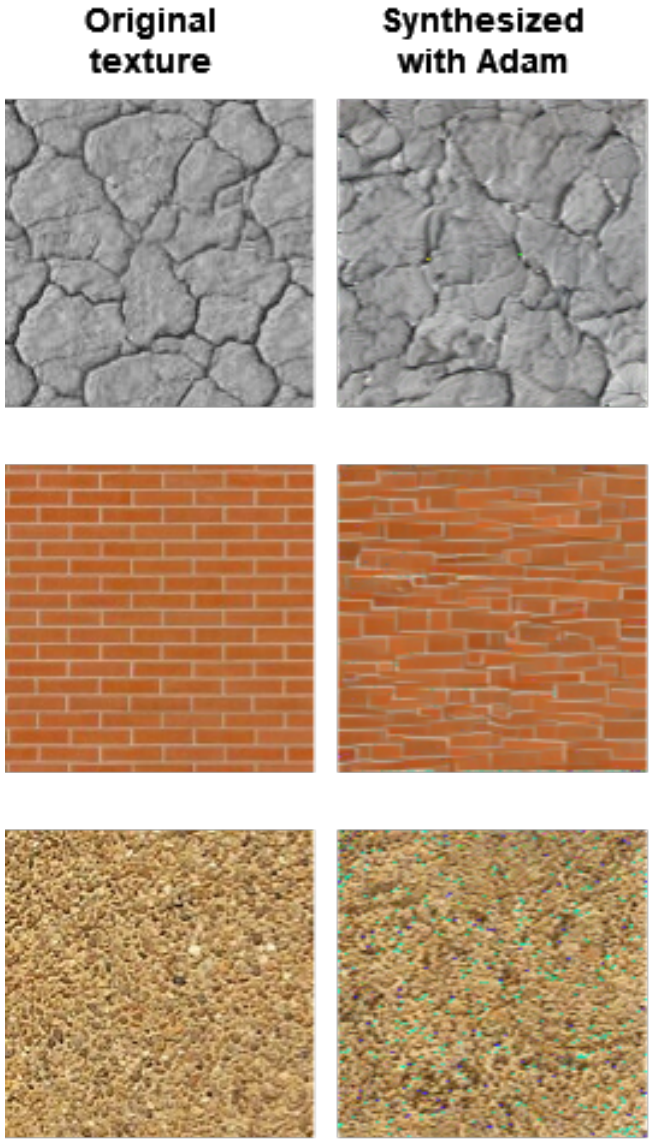

Figure 5.1: Synthesized textures (right) alongside their corresponding target textures (left). The synthesized textures exhibit a high degree of resemblance to their target textures, validating the efficacy of our texture synthesis approach.

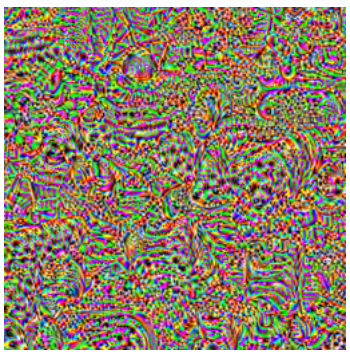

Figure 5.2: The noise image used in the experiment, designed to stimulate all channels and maximize the expressive capacity of the neural network and target texture.

5.2 Style Transfer

The style transfer experiment aimed to translate an artistic style from one texture to another, employing a similar methodology to the previously discussed texture synthesis experiment. By utilizing a real texture as the starting point, this experiment offered a fresh perspective and explored the interplay between different textures.

Through an iterative refinement process, we successfully achieved style transfer, effectively transforming the original texture to resemble the target texture. The resulting images exhibit a striking similarity to the target texture; however, depending on the texture used, remnants of the starting point can still be observed. Figure 5.3 visually demonstrates all the possible combinations obtained from the textures used in the previous experiment.

The favourable outcomes of the style transfer experiment provide strong validation for the effectiveness of our approach and underscore its potential for further advancement. The successful accomplishment of style transfer establishes a robust foundation for the subsequent phase of our research, particularly in the context of the reversibility experiment.

5.3 Reversibility

The results of the reversibility experiment demonstrate that the Runge-Kutta method exhibits a certain degree of reversibility; however, it falls short of achieving full reversibility. Complete reversibility is a necessary condition to ensure the existence of an invariant submanifold, which, in turn, enables a well-defined normalization.

The decision to refrain from decoupling the features can be justified by the observed loss of information during the adjustment process. When attempting to revert back, it becomes apparent that the original texture cannot be accurately recreated. The Runge-Kutta method does not effectively follow the trajectory defined by the gradient. If it did, both Runge-Kutta and Adam would not exhibit a similar level of information loss.

The conducted tests have revealed that the feature gradients, as anticipated, do not satisfy the Frobenius integrability condition. Consequently, they do not define a genuine invariant submanifold. This limitation prevents the application of normalization for layered decoupling.

Based on the obtained results, it can be concluded that achieving perfect reversibility is still a distant goal. Nonetheless, there is an improvement observed with the Runge-Kutta method in comparison to Adam. These findings suggest the necessity for further exploration into the potential contributions of numerical integration techniques, such as Runge-Kutta, in the context of synthesis and transfer tasks.

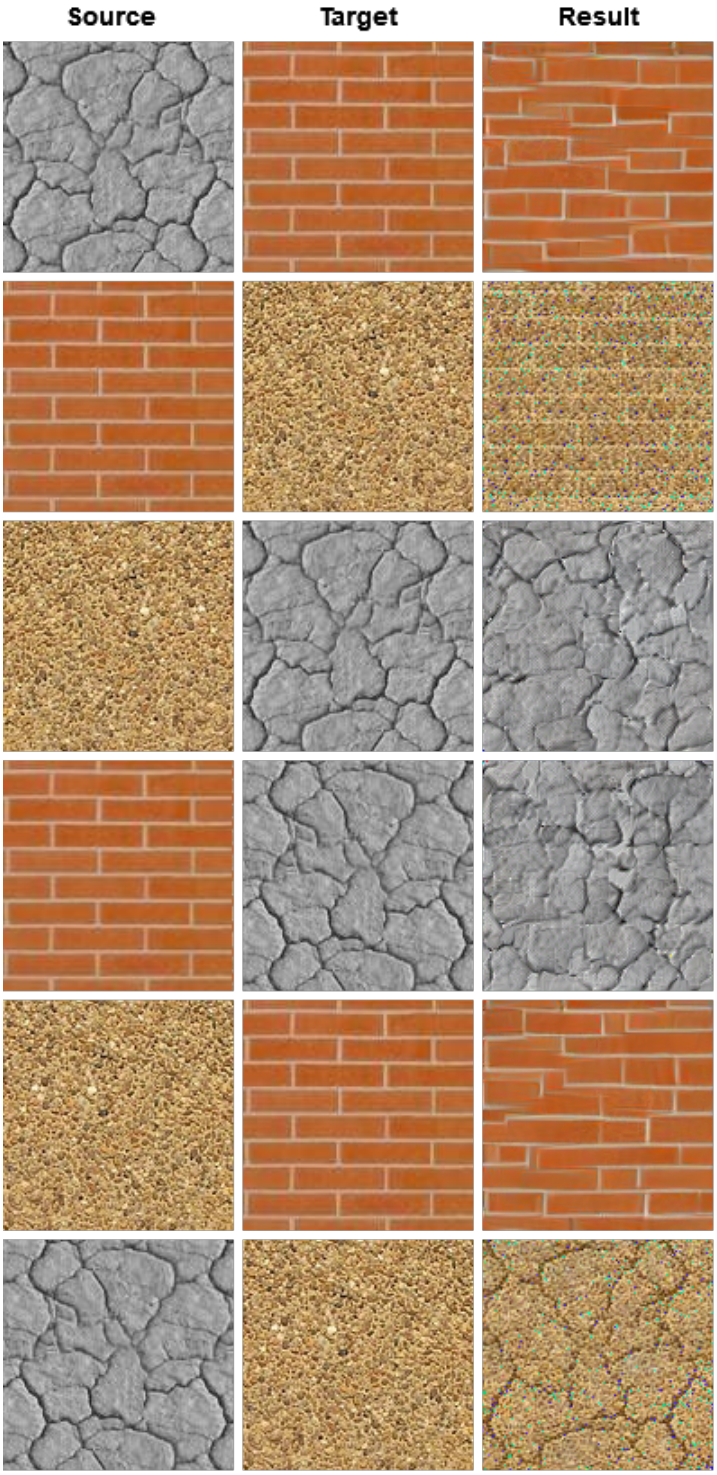

Figure 5.3: Results of the Style Transfer Experiment: Transformation of one texture's features into another's.

To visually illustrate the results, Figure 5.4 depicts the progressive transformation of the texture adjustment process in two examples, where a comparison is made between the Runge-Kutta method and Adam.

This limited reversibility poses a significant challenge, compounded by the similarity of the results obtained using Adam and the Runge-Kutta method. While a closer examination reveals a relatively lower degree of information loss in the Runge-Kutta version, the distinction between the two methods is more subtle than expected. Ideally, a clear differentiation between the two approaches would be preferable.

Figure 5.5 comprises three sub-figures, each providing valuable insights into the optimization process.

The first sub-figure showcases the deviation from the original texture, represented as the "error" using both Adam and Runge-Kutta methods. Initially, as the image undergoes transformation, the error rapidly increases until reaching a peak. In the subsequent phase, the image gradually returns to its original appearance, leading to a decrease in the difference. While Adam swiftly reduces the error due to its efficient optimization, it tends to fluctuate without making significant progress once it reaches a certain distance, indicating information loss. On the other hand, Runge-Kutta achieves a slower yet consistent reversal process that requires more iterations but steadily diminishes the difference. However, similar to Adam, it also encounters information loss and eventually plateaus without surpassing a certain threshold. These observations suggest that the global minimum for both methods is relatively close, with Runge-Kutta slightly lower.

The second sub-figure compares the loss computed by our cost function. The loss consistently decreases as the optimization process approaches the current target. The sudden jump near the beginning signifies a change in the current target, where the loss associated with it also changes. Similar to the previous sub-figure, Runge-Kutta exhibits a cautious and gradual progression, while Adam demonstrates a faster yet more erratic behaviour.

Lastly, the third sub-figure provides a direct comparison between the error and loss, which is particularly valuable due to the significant difference in the number of iterations employed by the two methods. This allows for a more meaningful and direct assessment. By analysing the graph from right to left, we can observe that both methods exhibit a similar trajectory and ultimately achieve a comparable final result, despite their distinct behaviours throughout the process.

In summary, while achieving perfect reversibility remains challenging, the experiment has provided insights into the potential advantages of using the Runge-Kutta method compared to gradient descent. These findings suggest the need for further exploration and testing of the Runge-Kutta method in combination with successful approaches from previous experiments.

5.4 Texture Synthesis and Style transfer with Runge-Kutta

In this section, we explore the use of the Runge-Kutta method as an alternative to the Adam optimization algorithm for texture synthesis and style transfer. The main goal is to investigate the effects of using Runge-Kutta on the adjustment process and its impact on the overall performance and feasibility of these tasks.

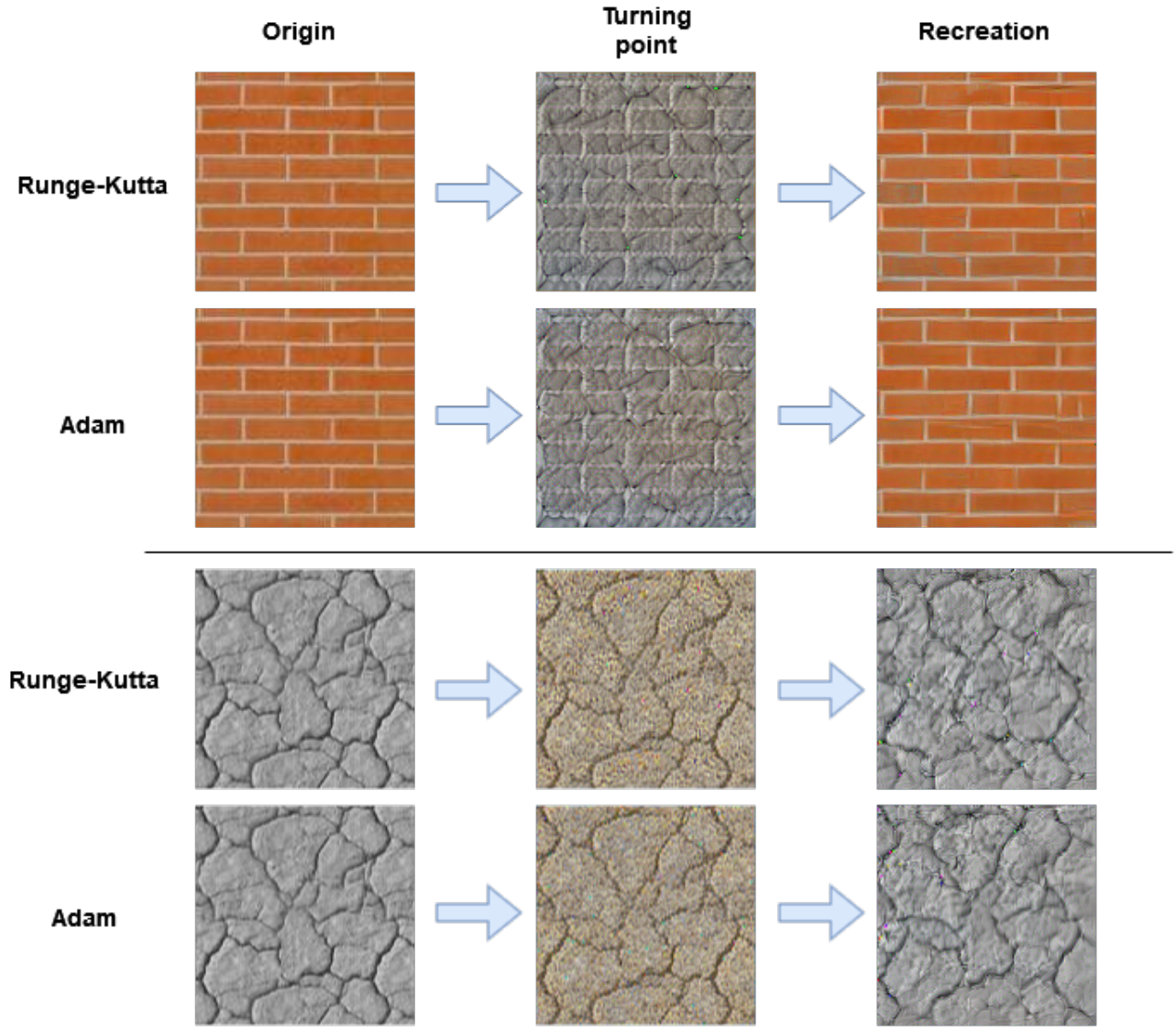

Figure 5.4: Results of the Reversibility Experiment: A comparison between Adam and Runge-Kutta. The start and turning point are consistent for both methods, with small differences observed in the recreated textures.

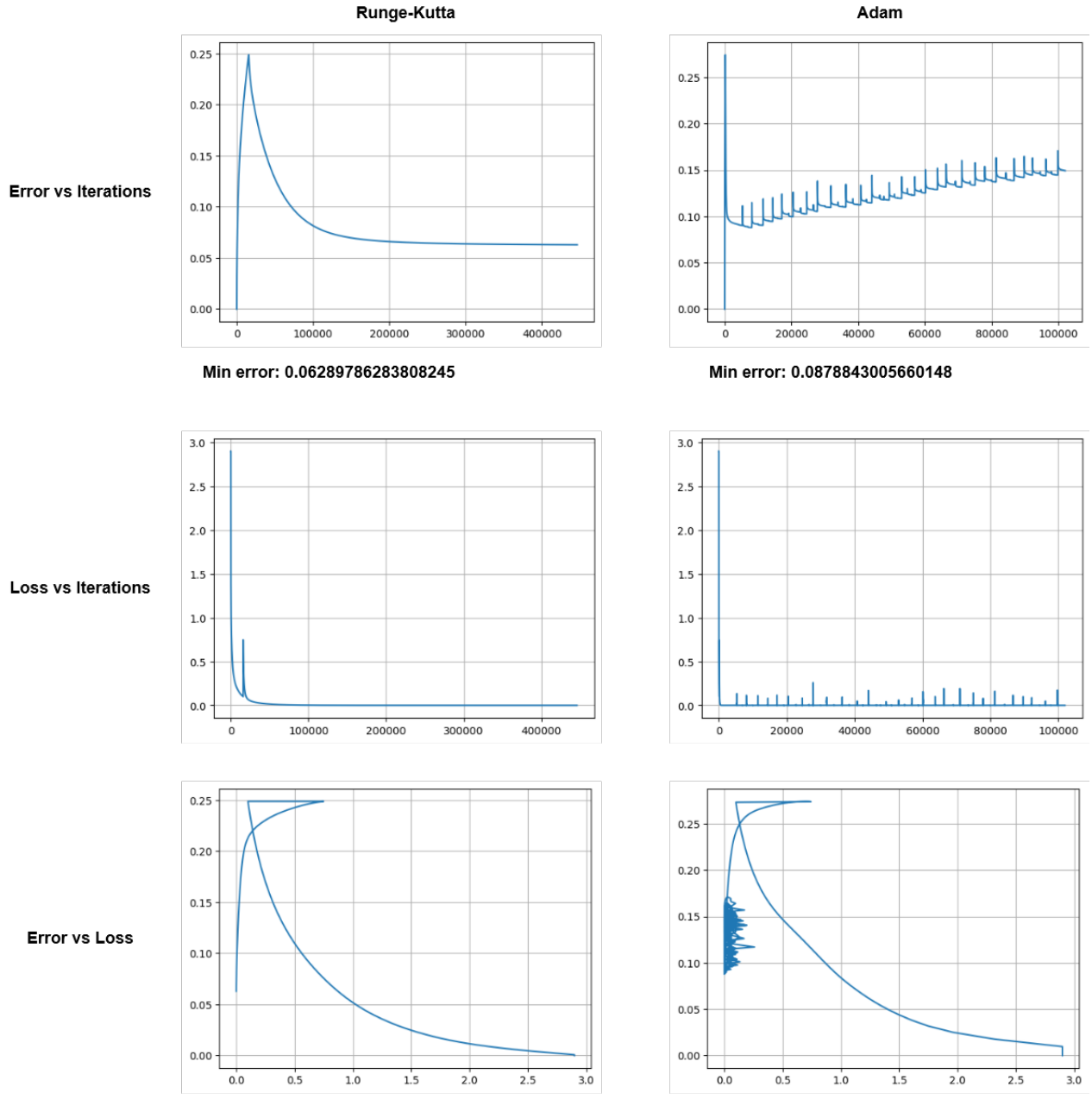

Figure 5.5: This figure consists of three sub-figures. The first sub-figure shows a comparison between Adam and Runge-Kutta for the error, representing the difference between the adjusted texture and the original base texture. The second sub-figure depicts the loss computed by our cost function for both methods. The third sub-figure compares the error and loss directly.

When applying the Runge-Kutta method, we observed a significant slowdown in the tuning process compared to Adam. This is primarily due to the optimization capabilities of Adam for finding function minimums and the higher computational cost associated with the Runge-Kutta method. This results in difficulties in effectively imposing the measured features onto the target texture. This slowdown poses challenges in achieving satisfactory results within a reasonable timeframe, making texture synthesis and style transfer infeasible in some cases. Figures 5.6 and 5.7 showcase the results of both texture synthesis and style transfer experiments and compares them with Adam. It can be observed that the results obtained with Runge-Kutta are inferior to those achieved with Adam, both in terms of visual quality and convergence speed.

The slower adjustment process of Runge-Kutta hinders the convergence to the desired target texture, leading to limitations in the quality of the synthesized or transferred textures.

In conclusion, the exploration of using the Runge-Kutta method for texture synthesis and style transfer reveals that it severely slows down the tuning process, making it impractical and infeasible in some cases. As a result, the Runge-Kutta method does not provide any notable practical advantages for these particular tasks. The increased computational cost impedes an effective imposition of measured features onto the target texture, hindering the achievement of satisfactory results within a reasonable timeframe.

5.5 Personal Reflections and Overall Conclusions

In this research, I explored the integration of nested normalizations with state-of-the-art texture modelling techniques to enhance the descriptive power of features for visual texture analysis and representation. My main objective was to investigate the feasibility of combining these approaches.

To achieve the objective, I developed a model that extracted spatial means from the channels of a pre-trained CNN network. I attempted to apply nested normalizations to decouple these features and enhance their descriptive capabilities. Through a comprehensive review of mathematical and theoretical concepts, I gained a thorough understanding of the underlying principles.

I successfully extracted meaningful features from textures and compared their spatial means using an effective method. The trajectory analysis of gradients provided insights into the temporal evolution and spatial variations of gradients during the style transfer process. I evaluated the performance of my proposed approach for style transfer and texture synthesis, identifying its advantages and disadvantages.

The results obtained from the experiments demonstrated the effectiveness of the proposed methodology in generating visually compelling textures and achieving style transfer. However, the challenge of achieving perfect reversibility remains. It is important to acknowledge the negative results encountered in this research, as they provide valuable insights and lessons learned.

While I succeeded in extracting high-quality features, the inability to decouple them represents a limitation of my work, while I have successfully explored the feasibility, the results are not what I had hoped for. This realization highlights the need for further advancements in this field to focus on creating a set of features that come

Figure 5.6: Comparison between the style transfer results. The visual quality of the Runge-Kutta exhibits variations, ranging from comparable to Adam, to significantly worse.

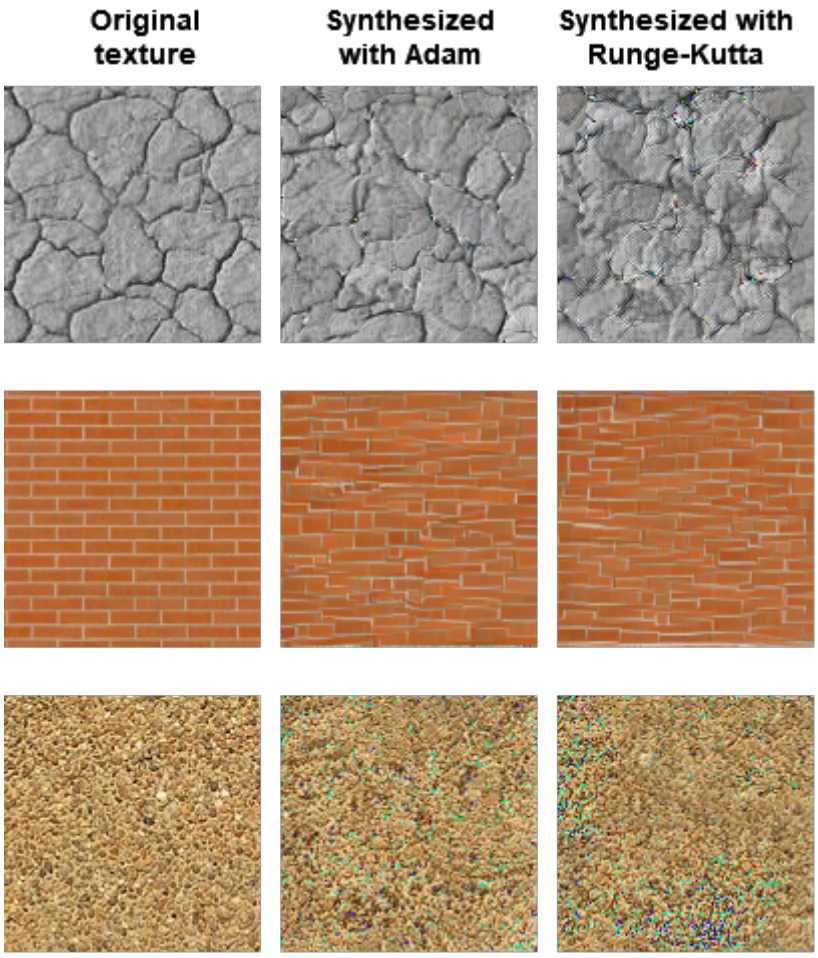

Figure 5.7: Comparison between the texture results achieved using Runge-Kutta and Adam. The visual quality of Runge-Kutta is now closer to that of Adam. However, it still falls short in terms of both visual quality and computational speed.

closer to meeting the requirements for decoupling, potentially by utilizing a more heavily modified CNN architecture.

Through this research, I have gained a deeper understanding of the complexities involved in texture modelling. I recognize the challenges and limitations of the current approach, including the non-meeting of theoretical requirements for deterministic decoupling. Nonetheless, I believe that the positive outcomes achieved and the potential for future exploration have laid a solid foundation for advancing this field.

In conclusion, this research has made significant progress in integrating nested normalizations with texture modelling techniques, providing valuable insights into their effectiveness and potential. It serves as a stepping stone for further advancements. By learning from the limitations of this work, future advancements can focus on refining feature decoupling and pushing the boundaries of texture modelling techniques. These efforts will contribute to the realization of more realistic and visually captivating results in the field of computer vision and image processing.

Bibliography

[1] Bela Julesz. "Visual pattern discrimination". In: *IRE transactions on Information Theory* 8.2 (1962), pp. 84–92.

[2] Stuart Geman and Donald Geman. "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), pp. 721–741.

[3] Javier Portilla and Eero P Simoncelli. "A parametric texture model based on joint statistics of complex wavelet coefficients". In: *International journal of computer vision* 40 (2000), pp. 49–70.

[4] Leon Gatys, Alexander S Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks". In: *Advances in neural information processing systems* 28 (2015).

[5] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2414–2423.

[6] Keyan Ding et al. "Image quality assessment: Unifying structure and texture similarity". In: *IEEE transactions on pattern analysis and machine intelligence* 44.5 (2020), pp. 2567–2581.

[7] Eduardo Martínez-Enrí quez, Maria del Mar Gonzalez, and Javier Portilla. "Deterministic Decoupling of Global Features and its Application to Data Analysis". In: *arXiv preprint arXiv:2207.02132* (2022).

[8] Eduardo Martínez-Enríquez and Javier Portilla. "Deterministic feature decoupling by surfing invariance manifolds". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 6049–6053.

[9] Javier Portilla and Eduardo Martínez-Enríquez. "Nested normalizations for decoupling global features". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 2112–2116.

[10] Olivier J Hénaff and Eero P Simoncelli. "Geodesics of learned representations". In: *arXiv preprint arXiv:1511.06394* (2015).

[11] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2008.

[12] Ricky T. Q. Chen. *torchdiffeq*. 2018. URL: `https://github.com/rtqichen/torchdiffeq`.