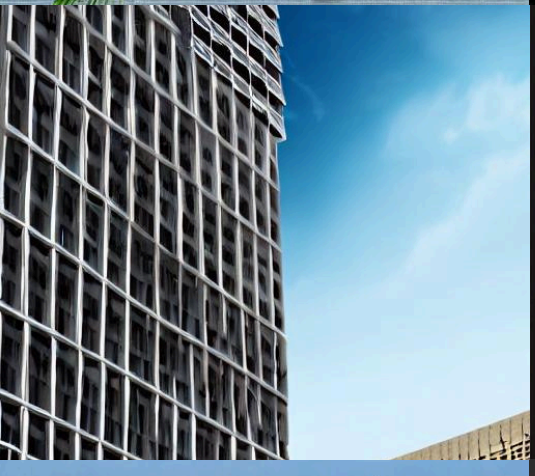
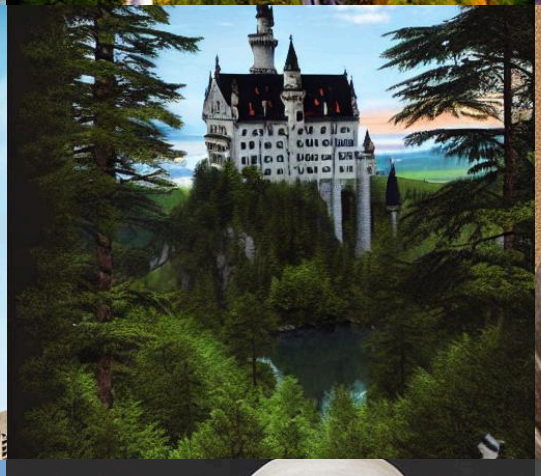
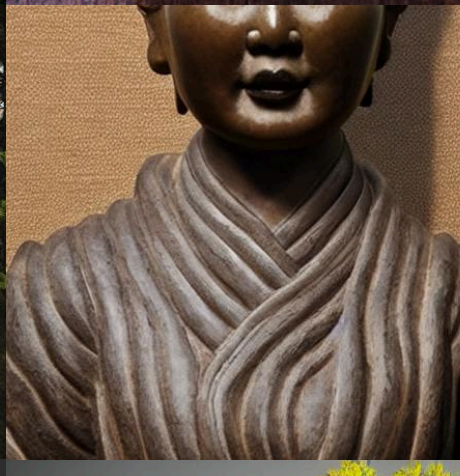
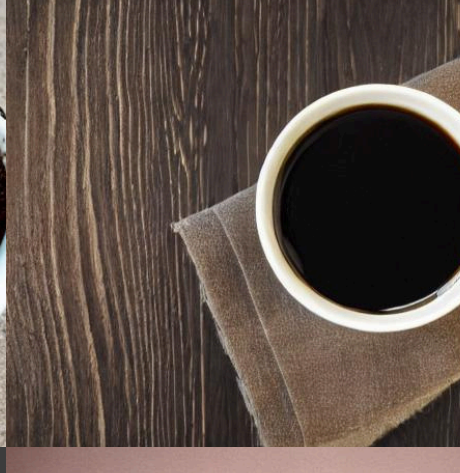









Shadows of Deception: Unveiling AI-Generated Images through Inconsistencies in Scene Lighting

Masterarbeit im Fach Artificial Intelligence

vorgelegt von

Muhammad Abdullah

geb. am 25.11.1995 in Kasur, Pakistan

angefertigt am

Department Informatik
Lehrstuhl Graphische Datenverarbeitung
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuender Hochschullehrer: Prof. Dr. Bernhard Egger, PD Dr. Christian Riess

Beginn der Arbeit: 20.12.2023

Abgabe der Arbeit: 24.06.2024

Erklärung

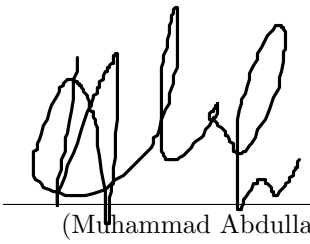
Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ich bin damit einverstanden, dass die Arbeit veröffentlicht wird und dass in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Graphische Datenverarbeitung, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Ich erkläre, dass ich ChatGPT nur zum Umformulieren einiger meiner eigenen Sätze und für Hilfe im Zusammenhang mit Latex verwendet habe. Ich habe weder ChatGPT noch ein anderes Large Language Model zur Generierung neuer Inhalte oder zum Umformulieren fremder Inhalte verwendet.

Erlangen, den 24.06.2024



(Muhammad Abdullah)

Contents

List of Abbreviations

AI Artificial Intelligence. 1, 3, 18

ARM Autoregressive Models. 1

BRDF Bidirectional Reflectance Distribution Function. 19, 20, 23

CLIP Contrastive Language-Image Pretraining. 10

CNNs Convolutional Neural Networks. 2, 6–8, 10, 12, 13, 18, 33

DDPM Denoising Diffusion Probabilistic Models. ix, 1, 2, 13, 15

DL Deep Learning. 1–3, 5, 8, 10, 18, 19, 25, 53

DNNs Deep Neural Networks. 5, 6, 10–13, 18, 19, 23, 38, 39

FBM Normalizing Flow-based Models. 1, 15

GAN Generative Adversarial Netowrks. 1, 2, 12, 13, 15, 16

GPU Graphics Processing Unit. 7

HiFiC High-Fidelity Generative Image Compression. 40

IID Intrinsic Image Decomposition. 22, 23

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 8

IR Inverse Rendering. ix, 1–3, 19, 22, 23, 33, 36, 53

LDM Latent Diffusion Model. 15, 28

LR Logistic Regression. 37

LTE Light Transport Equation. 19, 20

MLP Multi-Layer Perceptron. x, 6, 7, 13, 19, 37, 38

NLP Natural Language Processing. 8

ResNet Residual Neural Networks. 8

RF Random Forest. 37

SD Stable Diffusion. 26, 28–31, 42, 43, 45, 47–49, 51, 52

SGD Stochastic Gradient Descent. 5

SHC Spherical Harmonics Coefficients. 2, 3, 18, 19, 33–36, 41, 46, 50

SHL Spherical Harmonics Lighting. 2, 34

SVM Support Vector Machine. 37

VAE Variational Autoencoder. 1, 15

ViT Vision Transformer. 8, 10, 19, 37–39, 41, 50, 52, 53

VLM Vision Language Model. 26

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent advancements in Artificial Intelligence (AI) have made it very easy to generate photorealistic images. Now, anyone with a smartphone can create them within seconds by just describing the content of the image in natural language [1, 2, 3]. Due to their ease of generation, these AI-generated images are abundant across all social media platforms over the Internet [4]. On the positive side, these images can be benign, showcasing a unique blend of creative ideas. They can also help marketing companies to generate content quickly [5]. However, on the other side, they can also be used to spread misinformation, create deepfakes of celebrities for malicious purposes [6], and impersonate someone for fraudulent activities. Therefore, an automated pipeline to detect these images is necessary. It can help journalists, fact-checkers, law enforcement agencies, and governments to combat fake news and identity theft [7]. The recent techniques [8, 9] used to create AI-generated images differ from physically-based rendering systems [10] because they produce images without any knowledge of 3D scene geometry and illumination. This often leads to inconsistent lighting across various parts of the image. As humans are generally not good at detecting lighting inconsistencies [11], these inconsistencies typically go unnoticed. But with the recent advancements in Inverse Rendering (IR), these lighting inconsistencies can be detected to make an automated pipeline for detecting AI-generated images.

1.2 Challenges

Ideally, the detection pipeline should work to identify all AI-generated images irrespective of the method used to generate them. However, the rapid development of image generation techniques with the help of Deep Learning (DL) has fostered the development of image generation methods based on totally different architectures [12]. Different DL architectures that are used to create these generators include Generative Adversarial Netowrks (GAN) [9], Denoising Diffusion Probabilistic Models (DDPM) [8], Autoregressive Models (ARM) [13], Variational Autoencoder (VAE) [14], and Normalizing Flow-based Models (FBM) [15] etc. Images generated using these different generators exhibit different properties. For example,

GAN-generated images tend to show less diversity [16] and mimic the distribution of the training dataset, whereas DDPM produce more diverse images. However, GAN tends to produce high-fidelity images [16]. Similarly, images generated from different methods have different artifacts. These artifacts can appear either in the spatial domain or the frequency domain [17]. Often, generated images are subjected to further transformation *i.e.* compression and resizing when shared over different social media platforms [7]. Therefore, creating a detector that generalizes across various generators is challenging.

Since all DL-based image generators are trained directly in image space using a data-driven approach, they all exhibit lighting inconsistencies, regardless of their specific architecture. Therefore, our method of exploiting lighting inconsistencies can generalize across various generators. But to learn lighting inconsistencies, we model the properties of the 3D scene *i.e.* geometry, illumination from 2D images. This is inherently an ill-posed problem and often requires assumptions about the 3D scene to constrain the solution space. Recently, combining IR with DL, methods [18, 19, 20] have been proposed to learn the 3D scene properties from 2D images in a data-driven manner. Being data-driven methods, they face challenges in generalizing to real-world in-the-wild images.

Recently, various methods have been proposed to detect AI-generated images [7, 21, 17, 22, 23, 24, 25]. These methods can be categorized into three types [26]. The first type focuses on high-level semantic artifacts, such as facial asymmetries [27, 28], missing or inconsistent shadows [25, 29], and incorrect projective geometry [25]. The second type targets low-level artifacts, including artifacts in the frequency domain [17], and different fingerprints introduced by the specific architecture [30, 23, 31]. The third type of detector can be described as a data-driven detector, where the specific features learned to distinguish between real and generated images are not explicitly known. It includes detectors that use Convolutional Neural Networks (CNNs) [30, 22] to learn discriminatory features and detectors that use large pre-trained multi-modal networks like CLIP [21, 7]. These detectors generalize to unseen models, but when there are major architectural changes among image generators, performance drops substantially [22].

1.3 Contributions

We focus on semantic artifacts that analyze high-level scene characteristics. These include elements like shadows, lighting, and material properties such as reflections. In this work, we focus on exploiting inconsistent scene lighting in generated images and design a detector that tends to generalize on various image generators. Our detector is trained on estimated scene lighting of real and generated scenes. We use Spherical Harmonics Lighting (SHL) to represent scene lighting. SHL is a low-frequency approximation of scene's irradiance environment map [32].In SHL, Spherical Harmonics Coefficients (SHC) are used to represent scene lighting. To compute Spherical Harmonics Coefficients (SHC), we perform IR, which decomposes an image into 3D scene properties such as diffuse albedo, surface normals, and illumination. We train different binary classifiers to classify images into real or generated ones. To assess the generalization capabilities of our detector, we evaluate it on images generated by different

generators that were not included in the training data.
The structure of the report is as follows: Chapter 2 discusses recent advancements in DL that have led to the development of state-of-the-art AI image generators and various methods proposed by the research community to detect AI-generated images. It also outlines spherical harmonics representation of scene lighting and explains how IR can help us compute properties of the 3D scene from 2D images. Chapter 3 explains the process of creating a dataset of real and generated images used to train and evaluate the detector. Chapter 4 explains the experimental details incorporating IR pipeline, computation of SHC, and training of binary classifiers. Chapter 5 presents results, and chapter 6 summarizes the report and suggests future work.

Chapter 2

Fundamentals and Related Work

Image manipulation techniques have existed for a long time. One of the oldest image manipulation techniques is photomontage, where parts of different images are cut and combined to create a new composite image. Figure 2.1 shows a composite image made from 3 different images taken during the First World War. With the advent of digital image processing tools like Adobe Photoshop and GIMP, techniques such as morphing, inpainting, compositing, compression, and super-resolution have become commonplace in everyday life. These manipulations still demand familiarity with the specific image processing tool. With the recent advancements of DL, realistic-looking images can be created by just explaining the image content in natural language as a prompt. This section will dive deeper into the advancements of DL that led to developing image generators conditioned on natural language.

2.1 A Decade of Deep Learning

DL is based on computing features directly from the training data using Deep Neural Networks (DNNs). The term *deep* refers to multiple hidden layers in DNNs. These layers have trainable parameters or weights that are updated during the network training. Network training consists of two steps. The current weights are used to predict the output in the forward step. A loss function then calculates the error between the predicted and actual outputs. In the backward step, the gradient of the loss function with respect to each weight parameter is computed by propagating the error backward through the network. Then, an optimization algorithm like Adam [34] or Stochastic Gradient Descent (SGD) is used to compute the updated weights. This process, which contains forward and backward steps, is repeated multiple times for different image batches until loss converges. Network weights of trained DNNs act as important features computed directly from training data. These features can then be used for different downstream tasks. One downside of DNNs is that their features are not easily explainable and tend to overfit the training data. Training DNNs requires labeled data and substantial computational resources.

Figure 2.1: A composite photograph, originally known as "A hop over" constructed by Captain Frank Hurley, an Official War Photographer in WWI [33]

2.1.0.1 Multi-Layer Perceptron (MLP)

Two common types of DNNs are Multi-Layer Perceptron (MLP) and CNNs. MLP consists of multiple fully-connected layers. The term *fully connected* refers to the fact that each neuron (or perceptron) in a single layer is connected to every neuron of the previous layers. A typical MLP also contains an activation function after each fully connected layer to model the non-linearities of training data. ReLU, Sigmoid, and Tanh [35] are a few examples of standard activation functions. Recently, additional layers such as dropout[36] and batch normalization[37] have been introduced to address the issues of overfitting and training instability. Figure 2.2 shows an MLP with a single hidden layer without any extra activation or dropout layer.

2.1.0.2 Convolutional Neural Networks (CNNs)

The number of parameters for MLP explodes when applied to large dimensional inputs like images because of their fully connected nature. For images, CNNs works better [35]. A CNNs includes convolutional layers that utilize a convolution operation, where a small kernel or filter convolves over the image in a moving window fashion. Because of the smaller size of the kernel, the number of parameters is reduced significantly. The Sobel filter is an exemplary kernel where convolution with the image results in edges. Each convolutional layer contains

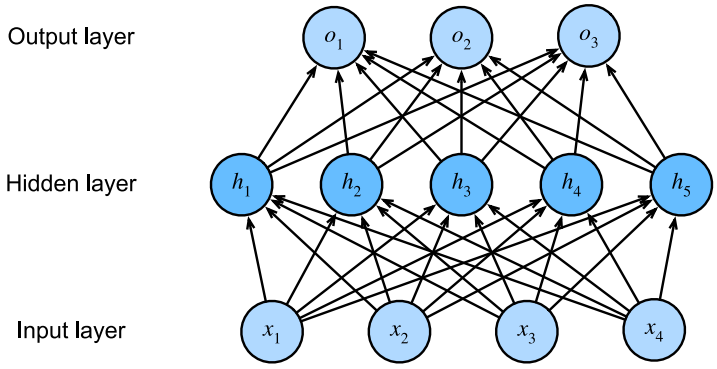


Figure 2.2: A MLP with a single hidden layer containing five neurons [35]

multiple kernels in CNNs, but these kernels are not predefined but learned directly from data. It should be noted that CNNs are translation-equivariant as the same kernel is convolved over the whole image [38]. If an object in the image is translated to another location inside the image, the resulting output of the convolutional layer will also translate equivalently. Translation-equivariance property helps them generalize better than MLP.

A typical CNNs contains additional layers as well. Each convolutional layer is followed by a pooling layer that reduces the spatial dimensions of the data and allows the network to look at a larger receptive field as we go deeper. Max-pooling and average pooling are common pooling operations [35]. In classification tasks, a small MLP is added, which takes image embedding at the last layer as input and computes probability over output classes using fully connected layers. Figure 2.3 shows the architecture of LeNet-5 [39], which classifies images of hand-written digits to the corresponding digits. Although LeNet-5 does not contain any normalization layer as the task is simple, more recent CNNs architectures contain normalization layers, e.g., batch normalization, to stabilize the training, which eventually helps them to converge faster [37].

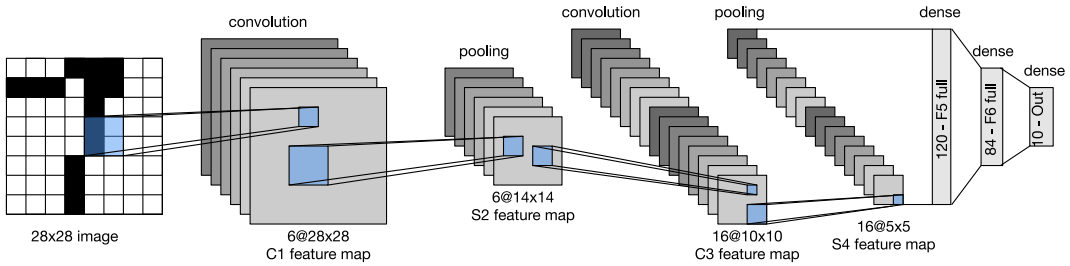


Figure 2.3: LeNet-5 architecture. Input is a hand-written digit, and output is probability over ten digits [35]

Although LeNet-5 was proposed in 1998, research in CNNs was still limited to a few researchers. Because training them on large amounts of data was computationally very difficult. Two things happened in the late 2000s. First, Nvidia released CUDA, a software interface that allows hardware-level parallelism for Nvidia proprietary GPU. The ImageNet [40] dataset was released in 2009, making a large amount of labeled data available for researchers. In 2012,

AlexNet, an architecture similar to LeNet-5, beat every other classifier to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenge. It sparked huge interest in research into CNNs. A key insight from AlexNet was the correlation between improved image representation and increased network depth. It led to the development of more deeper networks. ILSVRC challenge in 2014 was won by VggNet [41], which added more convolution layers and reduced the number of filters to keep the network parameters the same. Residual Neural Networks (ResNet) [42] was proposed in 2015, introducing a parameter-free identity shortcut. These identity shortcuts preserve the gradients from vanishing or exploding during backpropagation in deeper networks. ResNet is commonly used in many later architectures to compute image embedding.

2.1.0.3 Transformer

Another important DL architecture of the last decade is transformer [43]. It revolutionized sequence modeling tasks in Natural Language Processing (NLP). Figure 2.4 shows the transformer architecture. The transformer is based on an encoder-decoder architecture. The encoder models the dependencies between input tokens and projects them into a latent representation. The decoder receives encoder representations and predicts output tokens autoregressively. Autoregressive means each predicted token also depends on previously predicted tokens. Both the encoder and decoder employ scaled dot-product attention to model dependencies. The scaled dot-product is described in equation 2.1 where Q is the query matrix, K is the key matrix, V is the value matrix, and d_k is the dimension of the key vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

The attention mechanism in the encoder is called self-attention because Q, K, and V are initialized from the same input sequence, but in the decoder, where it is called cross-attention, Q comes from previous decoder prediction and K, V come from the encoder. The decoder also employs a mask that enforces the autoregressive property by allowing new tokens to get attended to previous tokens only. The encoder and decoder repeat the attention mechanism multiple times, which they call multi-head attention. These different attention layers run in parallel. Figure 2.7 shows the attention mechanism used in the transformer.

2.1.0.4 Vision Transformer (ViT)

Although transformer-based architectures have become the state-of-the-art for many language modeling tasks, their application in the vision domain was not thoroughly explored until 2021 when ViT was proposed [44], which contained only a transformer encoder. The core concept is to transform the image into 16x16 patches, flatten them, and handle them as sequences of tokens. An additional learnable embedding is introduced, and its state at the output of the transformer encoder acts as the global image representation [44]. An extra MLP is added at the end to classify images based on this global token [44]. The ViT architecture is shown in Figure 2.8.

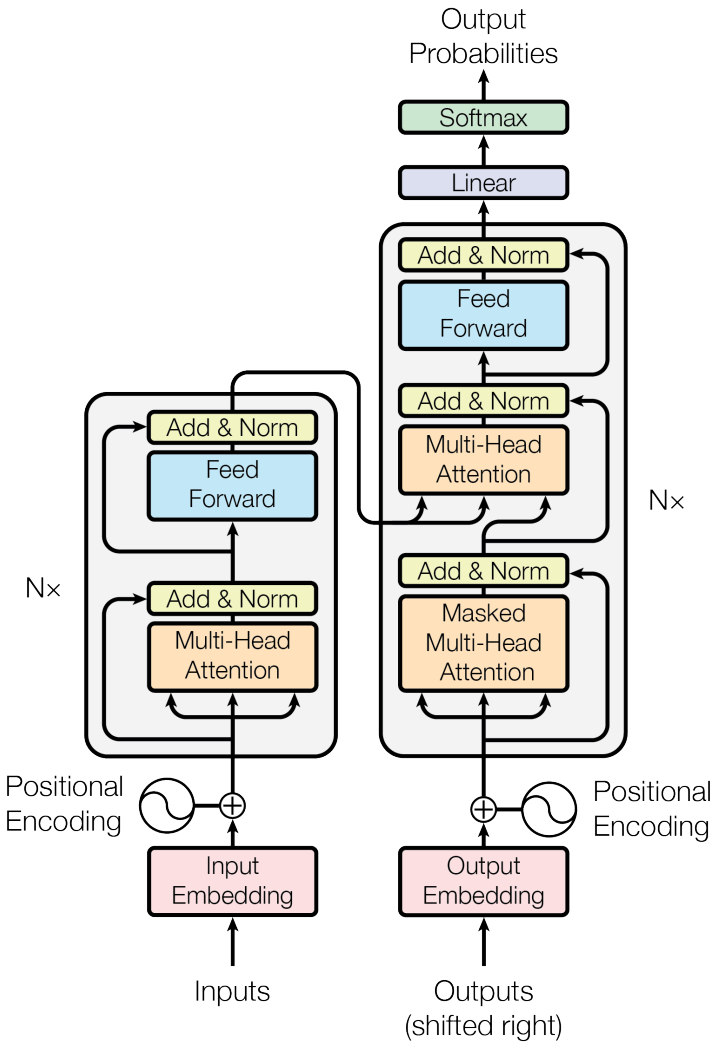


Figure 2.4: The transformer architecture [43]

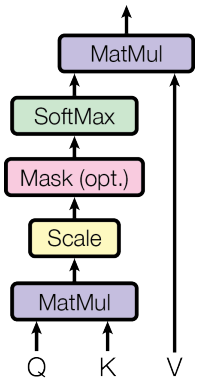


Figure 2.5: Scaled dot-product attention

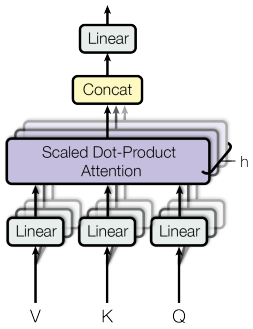


Figure 2.6: Multi-head attention

Figure 2.7: Attention mechanism in transformer [43] using query Q, key K, and value V

Theoretically, it's challenging for ViT to outperform CNNs because CNNs inherently learn inductive biases like translation equivariance and locality [44]. The lack of these inductive biases in transformer hinders its ability to generalize effectively with limited labeled data [44]. But if ViT is trained on larger datasets, the large-scale training beats inductive biases, and they can outperform CNNs [44].

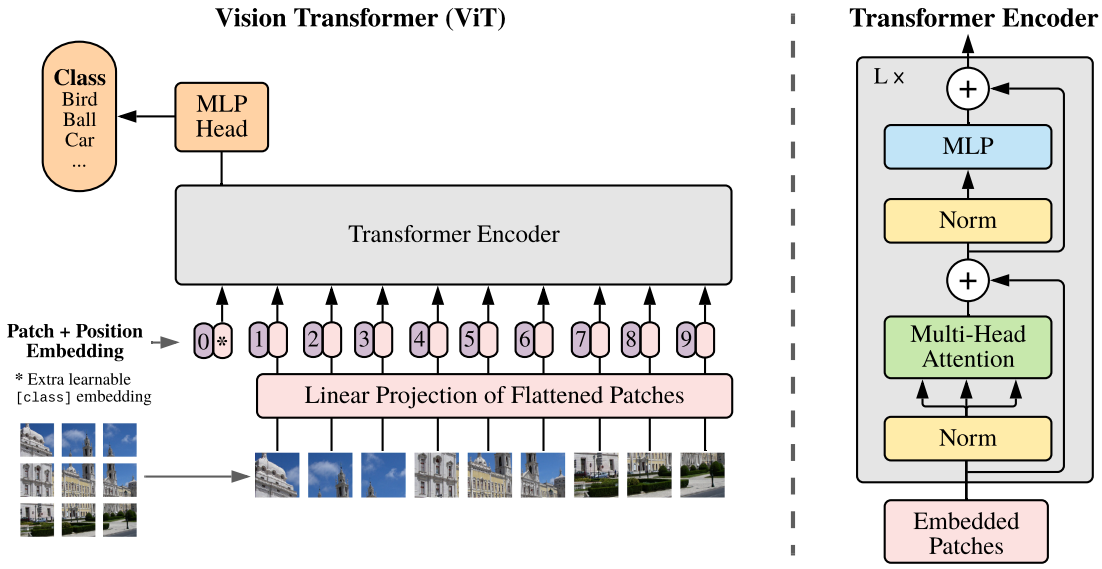


Figure 2.8: The vision transformer architecture [44]

2.1.0.5 Contrastive Language-Image Pretraining (CLIP)

As the DNNs are data-driven methods, they require high-quality training data. High-quality data is unavailable in many of the tasks *e.g.* medical imaging. An alternative to fully-supervised training is self-supervised pre-training. In Self-supervised learning, we try to learn intelligent representations from the vast amount of unlabeled data using a pretext task [45]. For example, the internet has billions of images with additional textual descriptions stored in *alt* attribute of the HTML *img* tag. Suppose we learn the joint vision-language representation by supervising the vision model with textual description and the language model with the image. In that case, we can use the learned representations on downstream tasks in a transfer-learning manner.

Pre-training methods have contributed significantly to the DL revolution of the last decade [46]. CLIP is one such pre-training network that is trained on 400 million image-text pairs collected from the internet [46]. CLIP architecture is shown in Figure 2.9. The image encoder is a ViT, and the text encoder is a hypernetwork [46, 47], which generates the weights of a linear classifier based on the text. A contrastive loss function maximizes the cosine similarity of text and image embedding for semantically similar pairs while minimizing the cosine similarity for dissimilar pairs. The learned representations can then be used in downstream tasks in zero-shot transfer learning, where we use the clip embedding without further fine-tuning, or in few-shot transfer learning, where we fine-tune clip embedding with a few examples of the

downstream task data.

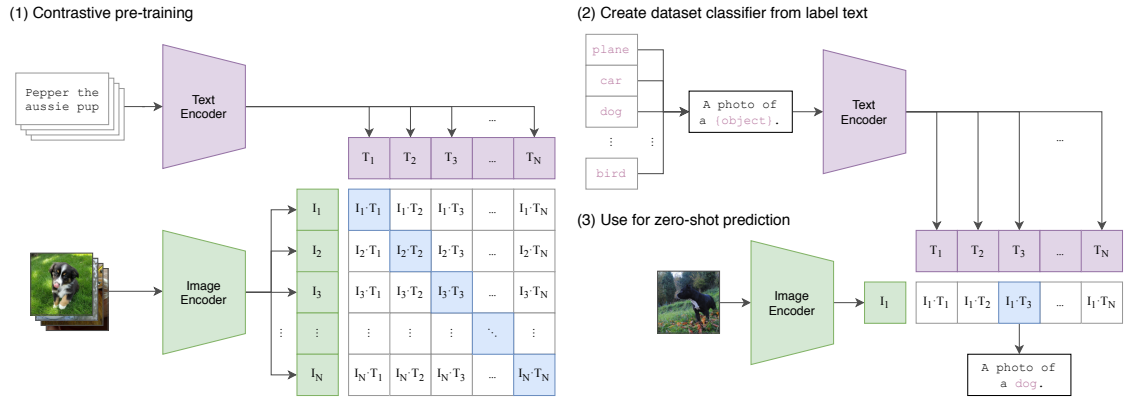


Figure 2.9: The CLIP pre-training and its use in a downstream classification task [46]

2.2 Overview of Image Generation Methods

DNNs have demonstrated outstanding performance on discriminative tasks such as classification, segmentation, and language translation mainly because of algorithms like backpropagation and the availability of high-quality training data. We will now discuss their application in generative tasks, such as image generation, which forms the basis for AI image generators. In discriminative tasks, we try to estimate the posterior $P(Y \mid X)$ directly from the training examples. In generative tasks, we try to learn the joint distribution $P(X, Y)$, and the posterior is estimated indirectly using Bayes' rule:

$$P(Y \mid X) = \frac{P(X \mid Y)\, P(Y)}{P(X)} \tag{2.2}$$

Since we have learned the joint distribution, we can sample it to create new data points. However, the difficulty lies in the fact that learning the joint distribution involves complex integrals to compute marginals, which often become analytically intractable when dealing with high-dimensional data. Approximate inference methods like Markov Chain Monte Carlo (MCMC) and variational inference are used, adding an additional complexity layer.

2.2.1 Evaluation Metrics

Before going into details of different algorithms, some evaluation metrics are presented here that are generally used to quantitatively assess the quality of generated images.

2.2.1.1 Inception Score

The inception score provides a metric to assess the quality of generated images, which correlates well with human evaluation [48]. Let x be a generated image, $p(y|x)$ is the conditional class distribution computed using a pre-trained image classification network, and $p(y)$ is the

marginal class distribution. If x contains a meaningful object, then the probability for a single class will be high, resulting in low entropy for $p(y|x)$. And if the generator produces diverse images, $p(y)$ will have high entropy. The inception score can be computed by taking the exponential of the mean relative entropy between $p(y|x)$ and $p(y)$:

$$\text{Inception Score} = \exp(\mathbb{E}_x[\text{KL}(p(y|x)||p(y)]) \tag{2.3}$$

As the inception score is based on class distribution, the generated samples must contain meaningful objects to get higher scores, and the generator should also generate images of diverse classes.

2.2.1.2 Fréchet Inception Distance (FID)

The inception score does not use the statistics of real-world samples to assess the quality of generated images [48]. Instead of using class distribution from a pre-trained classification network, the FID score [49] computes the variability of generated samples from the real-world samples by comparing their image embedding computed using a pre-trained CNNs. The mean vector and covariance matrices for these embedding are computed. The FID score can be computed using the following equation where μ_r and μ_g are the mean image embedding of real and generated images, C_r and C_g are the covariance matrices of real and generated images embedding, $|\cdot|_2^2$ represents the squared Euclidean distance, and $\text{Tr}(\cdot)$ denotes the trace of a matrix:

$$\text{FID} = |\mu_r - \mu_g|_2^2 + \text{Tr}(Cr + C_g - 2(C_r C_g)^{1/2}) \tag{2.4}$$

Unlike the inception score, a lower FID score indicates better similarity of generated images to the real-world samples.
After discussing metrics used to evaluate the quality of generated samples, we now discuss the architectural details of different DNNs-based image generators.

2.2.2 Generative Adversarial Networks

GAN was the first DNNs-based architecture that showed significant improvement in generating realistic images. GAN learns the approximate inference from training data by simultaneously training two DNNs. One is called the generative model (G), and the other is called the discriminative model (D). G learns the data distribution, and D predicts the probability of the generated sample being a real sample from the training data or a fake sample generated from G, and both work in a minimax manner to minimize each other's gains [9]. G tries to generate samples that D cannot distinguish from real samples, while D tries to improve itself to always correctly identify real and generated samples. A unique solution is obtained when G is so good at generating samples that D cannot predict if it is real or generated and predicts a probability of 0.5 [9]. D and G objective functions are given in equations 2.5 and 2.6, respectively, where x represents samples from training data, and z represents samples from random noise and the overall training objective is to maximize the D's objective function using gradient ascent and minimize the G's objective function using gradient descent:

Discriminator Objective Function:

$$\max_D V(D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.5}$$

Generator Objective Function:

$$\min_G V(G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.6}$$

The original GAN paper used MLP for both G and D [9], but later architectures such as deep convolutional generative adversarial networks (DCGANs)[50] introduced CNNs architectures where learned features in D and G can be used to model general image representations and can be used in novel downstream tasks. The GAN architecture is depicted in figure 2.10.

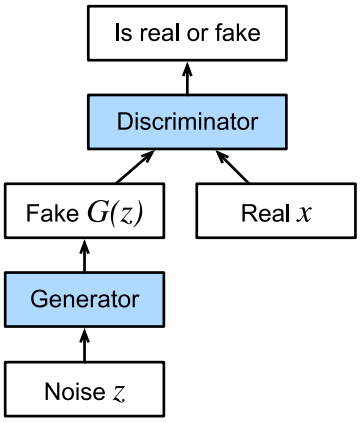


Figure 2.10: The GAN training mechanism [35]

The original GAN architecture generates samples unconditionally, but with a small modification, we can generate samples of a particular class. ConditionalGAN [51], shown in Figure 2.11, passes extra class embedding to both G and D. In this way, G and D learn class-conditioned distributions $G(z|y)$ and $D(x|y)$ respectively.
A common issue with GAN is mode collapse, where G only generates a limited variety of samples every time. This happens because G knows of a few samples that D misclassifies and generates them repeatedly. To address this problem, a careful balance in the training of G and D is required. In Figure 2.12, we show a few image samples generated from ProGAN [52], which significantly enhances the generated image quality by countering unhealthy competition between G and D [52].

2.2.3 Denoising Diffusion Probabilistic Models

The DDPM[8] is another class of DNNs-based generative models that produce high-quality images. Unlike GAN, they do not have a competing generator and discriminator. Their working principle is based on autoregressive denoising. They contain a forward process where Gaussian noise is added to the image and a backward process where noise is estimated using a DNNs (U-Net) and used to compute the resulting noise-free image. This backward process is repeated multiple times until we get the original image. Figure 2.13 depicts these forward and

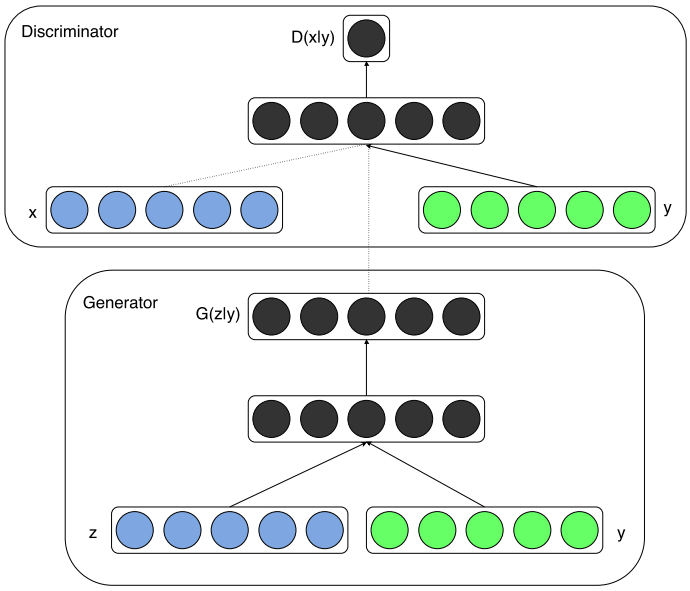


Figure 2.11: Conditional GAN architecture [51]

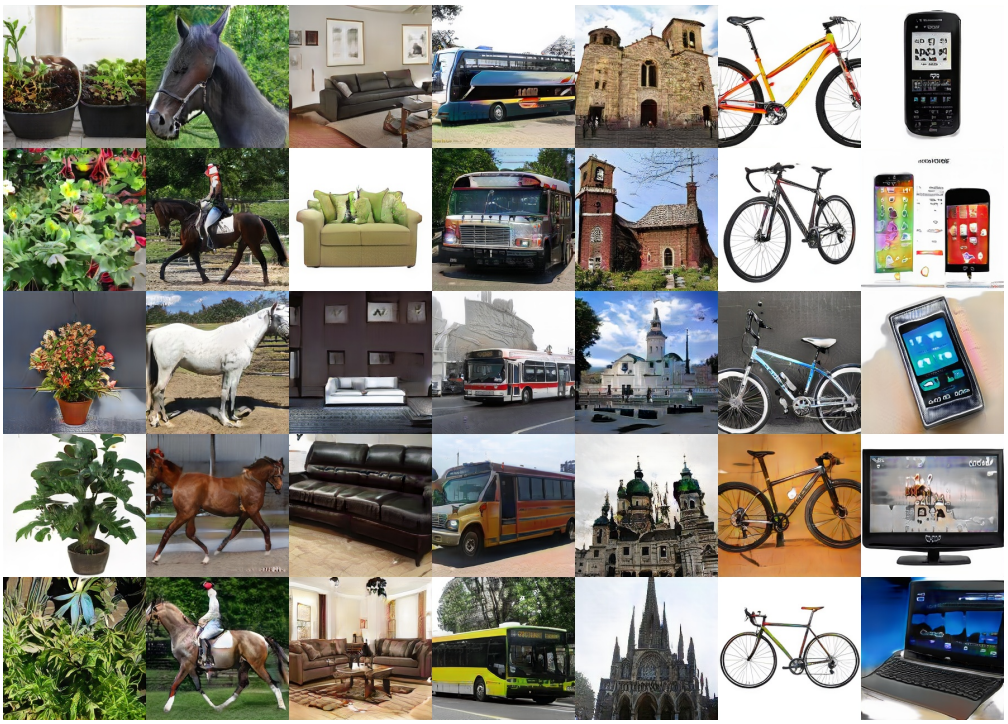


Figure 2.12: Some ProGAN generated images [52]

backward processes. The forward process $q(x_t|x_{t-1})$ forms a Markov chain with pre-calculated mean and variances for each step. There are different variance schedulers that are used during training. The backward process denoted as $p_\theta(x_{t-1}|x_t)$ forms a parameterized Markov chain [8] where these parameters are learned during training. As both $q(x_t|x_{t-1})$ and $p_\theta(x_{t-1}|x_t)$ are first-order Markov chains, we can estimate their joint distribution $q_\theta(\mathbf{x}_{1:T})$ and $p_\theta(\mathbf{x}_{0:T})$ using the equation 2.8 and 2.10 respectively, where β_t is the variance at step t during the forward process, μ_θ and Σ_θ are learned mean and variances during the backward process:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2.7}$$

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{2.8}$$

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2.9}$$

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \tag{2.10}$$

Training is performed by optimizing the variational bound on negative log-likelihood [8]:

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \tag{2.11}$$

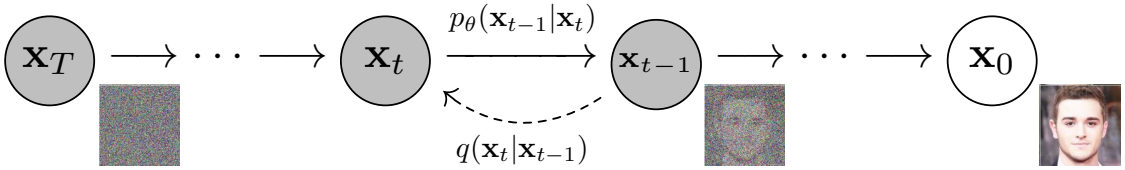


Figure 2.13: The probabilistic graphical model for DDPM [8]

As the DDPM applies the forward and backward processes in image space, they are computationally costly [53]. Alternatively, the Latent Diffusion Model (LDM) [53] first transforms the image into a latent space using the encoder of VAE and then performs the diffusion process there, resulting in fewer computations. Finally, the generated sample in latent space is converted back to image space using the decoder of VAE. The LDM also introduced cross-attention layers in the architecture, which can be used for general conditioning inputs such as text or images [53]. The architecture of LDM is shown in Figure 2.14.
Other than GAN and DDPM, VAE [14], Normalizing Flow-based Models [15], and Energy-based Models [54] can also be used to generate images. However, their adoption remains lower compared to GAN and DDPM. Therefore, many detection methods only focus on GAN and DDPM generated images.

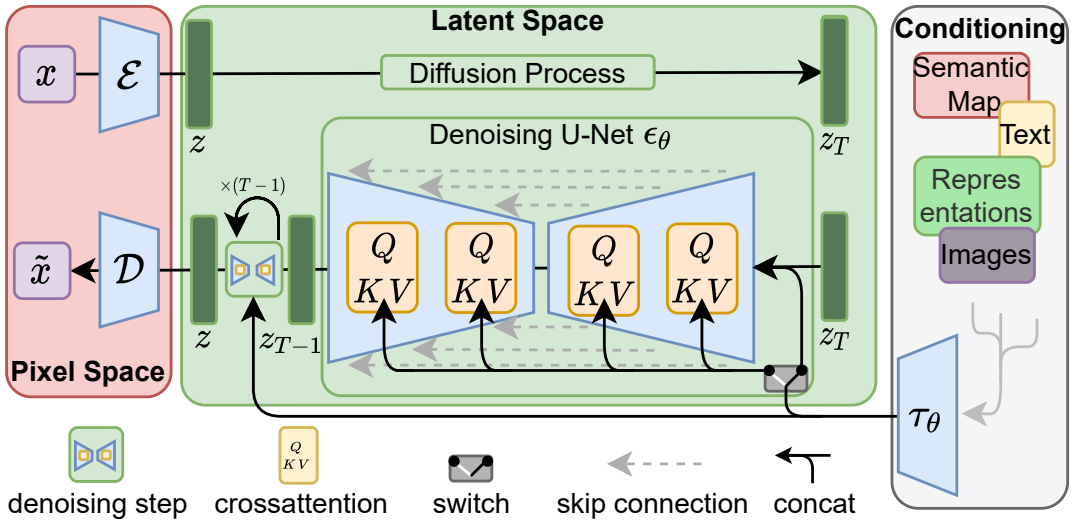


Figure 2.14: The latent diffusion model architecture [53]

2.3 Detection of AI-Generated Images

The research community has proposed various methods for detecting AI-generated images targeting high-level semantic features to low-level imperceptible artifacts. These detection methods are typically evaluated on different datasets, making it difficult to claim the superiority of one method over the other.

We use the taxonomy of detection methods created by Tariang *et al.* [26], depicted in Figure 2.15, where detection methods have been categorized into three categories. The first category contains methods targeting high-level semantic inconsistencies such as inconsistent shadows and lighting. The second category consists of methods that target low-level features such as artifacts introduced by the AI-image generator itself and the absence of noise added by image capturing devices *i.e.* camera. The third category contains data-driven methods where discriminant features between real and generated images are learned directly from data and cannot be fully explained. In this section, we will begin by exploring detection methods that focus on low-level artifacts, then examine data-driven approaches, and conclude with methods that identify high-level features most relevant to our work.

Speaking of low-level artifacts, Bi *et al.* [55] argue that upsampling operations in conditional GAN change the noise characteristics. They use a denoising network to estimate the noise. Then, they take the discrete Fourier transform of the noise and compute the average of the amplitude spectra. They use this average value to classify real and generated images. Sinitsa *et al.* [23] also follow a similar direction. For a given number of images generated from the same generator, they compute image residuals using a denoising filter and try to learn a fingerprint that correlates with these residuals. They use this fingerprint to detect if unseen images are generated from the same generator. Zhang *et al.* [56] follow a different direction. For a given number of real and DDPM-generated images, they follow the inverse diffusion process and compute noise at each step. They call it *diffusion noise features*. They use these

diffusion noise features to classify real and generated images.

Considering purely data-driven approaches to detect AI-generated images, Ojha *et al.* [21] argues that using pre-trained CLIP image embeddings, real images can be differentiated from generated images. As CLIP is trained on a large number of real images, it maps real and generated images to a different latent representation. However, to generalize across different generators, classifiers with linear decision boundaries are preferred, as non-linear classifiers try to overfit the training set. Ju *et al.* [57] first use ResNet50 to compute image embedding for the whole image. Then, they use a unique Patch Selection Module (PSM) to select important patches. They compute patch embeddings using ResNet50. Then, they fuse the global and local features and pass them to a classifier to detect real and generated images. Cozzolino *et al.* [58] use transfer learning to quickly adapt to the target domain of unseen generators. First, given a set of real and generated images from a source generator, they train an auto-encoder by reconstructing the samples. The latent space of the auto-encoder is then used to classify real and generated samples. They fine-tune their auto-encoder for a new unseen generator with a few samples from the target domain. They argue that the latent space preserves all the information required to predict real and generated images.

The next section presents a review of methods that target high-level semantic artifacts, which are closely related to our proposed method.

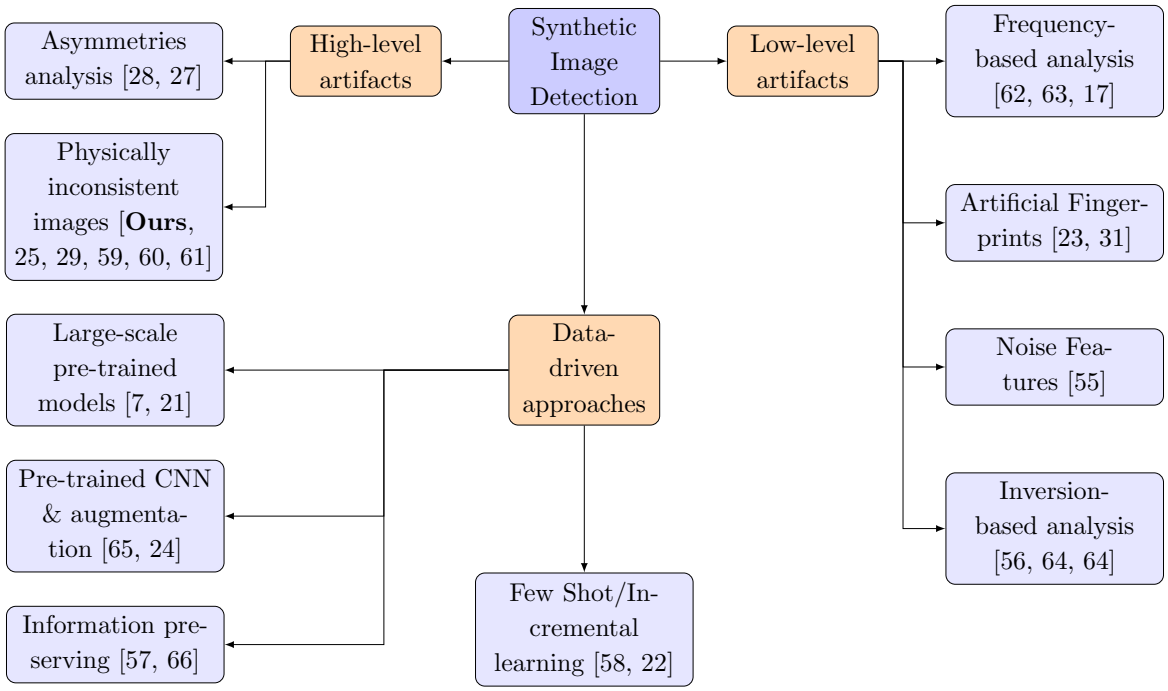
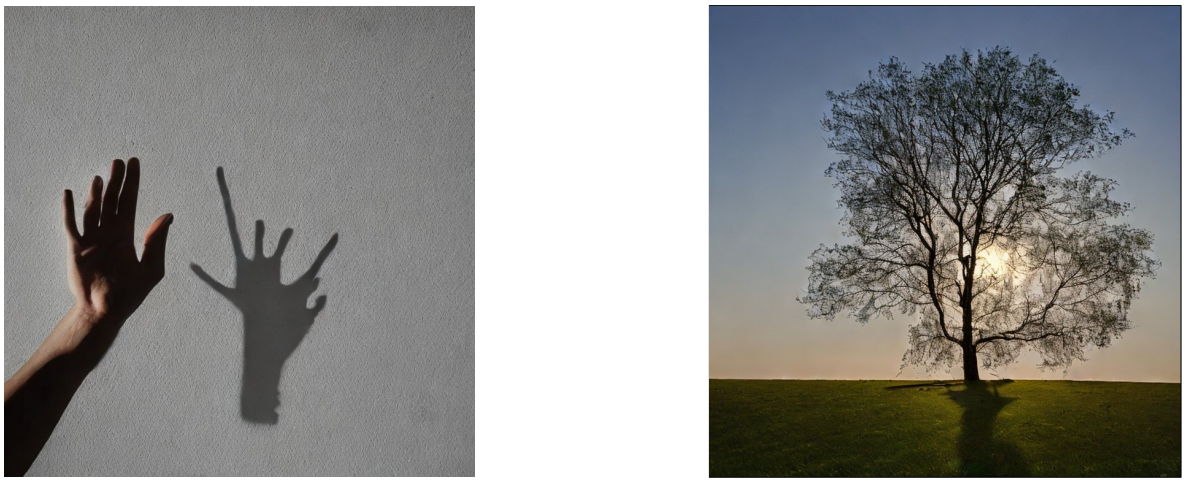


Figure 2.15: Classification of different approaches used for detecting AI-generated images, based on Tariang *et al.* [26]

2.4 Using Scene Lighting As a Discriminator

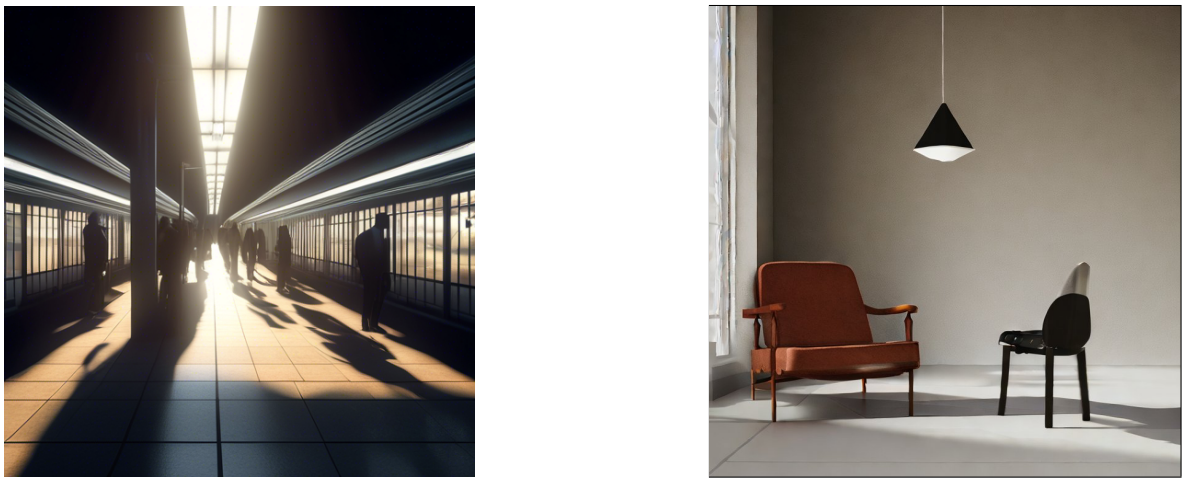
DL based image generators are different from physically based rendering systems. They are generally trained directly in image space or latent representation of image space without knowing anything about scene geometry, camera pose, lighting direction, and material properties. They try to learn scene properties implicitly from the training data. The loss functions used to train these DL image generators do not enforce any of the intrinsic properties of the scene, so they may produce physically inconsistent images. For example, images with physically inconsistent shadows, inconsistent lighting, or inconsistent material reflectance properties (absence of specular highlights on glossy surfaces). A few exemplary AI-generated images with physically inconsistent shadows are shown in Figure 2.16. Figure 2.17 shows a few generated images with inconsistent lighting.



(a) **(b)**

Figure 2.16: A few exemplary AI-generated images with physically inconsistent shadows. In (a) the shadow of the hand is not correct, (b) the tree trunk and its shadow do not match. These images are generated from the Stable Diffusion v1.5.

Inconsistent shadows and lighting have already been studied to detect image forgeries. Kee *et al.* [29] combine different constraints using cast shadows and attached shadows to find the projected location of the point light source. Inconsistent shadows result in the inability to find a solution using computed constraints. Sarkar *et al.*[25] use DNNs to predict binary masks for semantic object and shadow. They pass these predicted shadow-object masks to CNNs based binary classifier to classify them into real or non-real. Johnson *et al.* [67] tries to estimate the direction of the light source in outdoor scenes, and by exploiting the inconsistencies in light source direction, image manipulation can be detected. Riess *et al.* exploit inconsistencies among the color of the light source in different image regions to detect manipulations. Johnson *et al.* [60] models complex lighting environment with Spherical Harmonics Coefficients (SHC) and by using inconsistencies among SHC on different semantic parts of the image, image forgeries can be detected.
Our work is an extension of the work done by Johnson *et al.* [60] with some modifications



(a) **(b)**

Figure 2.17: A few exemplary AI-generated images with physically inconsistent lighting. In (a), we have powerful lighting at the top, but some of the objects (pillar on the left side) are not illuminated, (b) The effect of the light source at the top is not consistent on the floor. These images are generated from the Stable Diffusion v1.5.

and improvements. Unlike Johnson *et al.* [60] where they assume constant reflectance and manually compute surface normal along the occluding boundary, we run an IR pipeline to estimate the reflectance (diffuse albedo) and surface normals. Rather than computing SHC for green channels only, we compute them for red, green, and blue channels. We use DL based DNNs (MLP, ViT) to learn inconsistencies in SHC of complex indoor and outdoor scenes.
In the following section, we discuss how complex lighting environments can be approximated using SHC and how IR can be applied to compute intrinsic scene properties *e.g.* surface normal, and diffuse albedo.

2.4.1 Approximating Irradiance Environment Map with Spherical Harmonics

In real scenes, the light-matter interactions depend on the material and geometry of the objects present. As the light rays come from all directions, we must account for interactions for all light rays to accurately model the scene lighting. Fortunately, these interactions generally result in an equilibrium state that we can use to represent the scene lighting. In computer graphics, this equilibrium state is described by the Light Transport Equation (LTE) [10]. The LTE models the total reflected radiance of a point on an object in the scene in terms of emissions from that point, reflectance properties of the object, and the distribution of incoming lighting on that point [10]. The LTE is described in the following equation, where L models the reflected radiance from point p, ω_o indicates the viewing direction, ω_i indicates the incoming light direction, f indicates the Bidirectional Reflectance Distribution Function (BRDF), $(t(\mathbf{p}, \omega_i)$ refers to the point from where light is coming and θ_i is the angle between ω_i and surface normal at point p:

$$L(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_{S^2} f(\mathbf{p}, \omega_o, \omega_i) L(t(\mathbf{p}, \omega_i), -\omega_i) |\cos\theta_i| \, d\omega_i \tag{2.12}$$

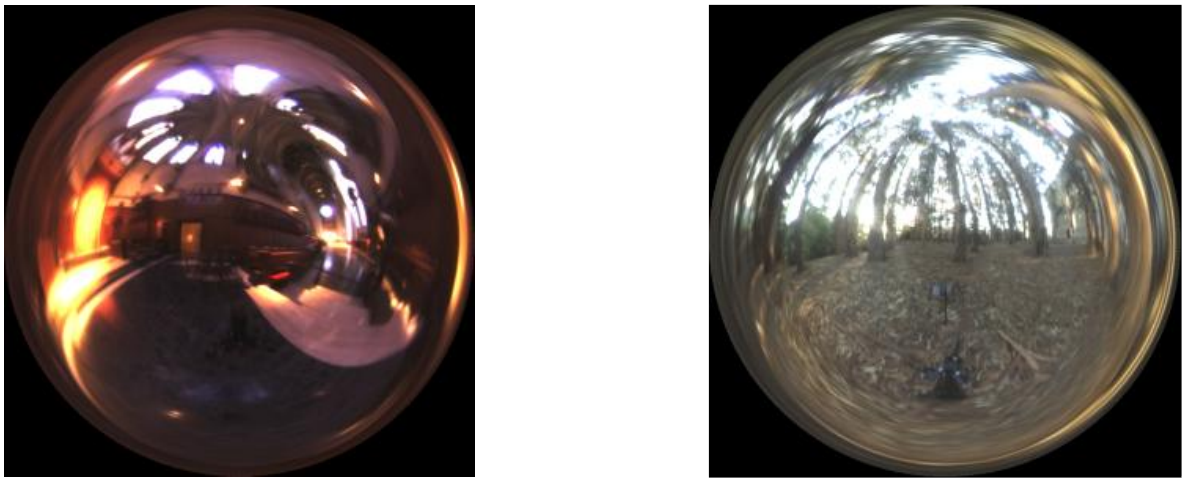
Evaluating the LTE analytically is difficult as incoming light on a point also depends on all other objects in the scene [10]. Approximate methods *e.g.* Monte Carlo integration can be used to solve the expression for approximate results [10]. Alternatively, we make some assumptions about the scene and simplify the LTE. For example, if the BRDF is Lambartian, we can move the BRDF outside of the integral, and if a distant light source illuminates the scene, L_e can be ignored. The resulting LTE can be represented as Equation 2.13 where $\rho(\mathbf{p}) = f(\mathbf{p}, \omega_o, \omega_i)$ is diffuse albedo, and $\cos\theta_i$ is replaced with $L.N_p$ where $dot(.)$ indicates the dot product:

$$L(\mathbf{p}, \omega_o) = \rho(\mathbf{p}) \int_{S^2} L(t(\mathbf{p}, \omega_i), -\omega_i) |L.N_p| \, d\omega_i \tag{2.13}$$

It can be seen that the integral is now accumulating the incoming light from all directions *i.e.* irradiance. We denote the irradiance as $E(N_p)$, and it is a function of surface normal only at point p [32]:

$$E(N_p) = \int_{S^2} L(t(\mathbf{p}, \omega_i), -\omega_i) |L.N_p| \, d\omega_i \tag{2.14}$$

As the irradiance is a function of surface normal only, it can be approximated by placing a metallic ball in the scene. The resulting scene map on the metallic ball is called the irradiance environment map. Figure 2.18 shows a few examples of the irradiance environment maps.



(a) **(b)**

Figure 2.18: A few exemplary irradiance environment maps taken from [68]

As the irradiance environment map is defined over a sphere's surface, we can approximate it using spherical harmonics. The spherical harmonics represent 2D functions defined over a surface's sphere as an infinite sum of weighted spherical harmonic functions [69]. It is similar to the Fourier series, where a 1D signal is represented as a weighted sum of sinusoids. The weights in spherical harmonics representation are called spherical harmonics coefficients.

$E(N_p)$ is represented as spherical harmonics in the following equation where c_l^m are spherical harmonics coefficients and $Y_l^m(N_p)$ are spherical harmonics basis functions:

$$E(N_p) = \sum_{l,m} c_l^m Y_l^m(N_p) \tag{2.15}$$

Here, $l \in [0, \infty]$ and $m \in [-l, l]$ are called order and degree, respectively. A mathematical representation of $Y_l^m(N_p)$ is given in the following equation where N_p is represented in polar coordinates $N_p \rightarrow (\theta, \phi)$, K_l^m is the scaling factor and P_l^m are the Legendre polynomials [69]:

$$Y_l^m(\theta, \phi) = \begin{cases} \sqrt{2} K_l^m \cos(m\phi) P_l^m(\cos\theta), & \text{if } m > 0 \\ \sqrt{2} K_l^{-m} \sin(-m\phi) P_l^{-m}(\cos\theta), & \text{if } m < 0 \\ K_l^0 P_l^0(\cos\theta), & \text{if } m = 0 \end{cases} \tag{2.16}$$

Ramamoorthi *et al.* [32] derive the Y_l^m for irradiance environment maps and show that, for Lambartian surfaces, they can be approximated with spherical harmonics up to the order 2 [32], which results in only nine spherical harmonics basis functions and spherical harmonics coefficients required for computing irradiance environment maps. The spherical harmonics representation of a few exemplary environment maps are given in Figure 2.19.

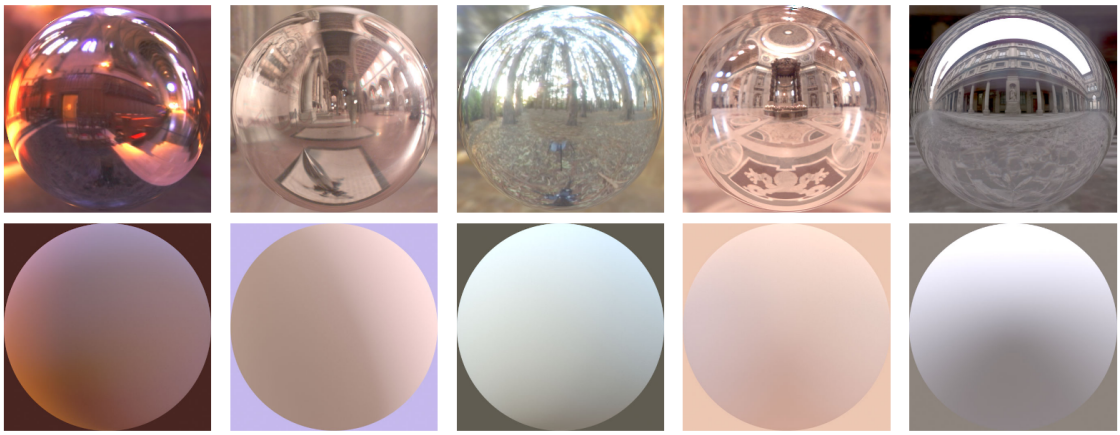


Figure 2.19: The spherical harmonics representation of a few exemplary irradiance environment maps (top row) rendered on surface's sphere (bottom row) [60]

We can now rewrite the total reflected radiance again with the spherical harmonics representation of irradiance:

$$L(\mathbf{p}, N_p) = \rho(\mathbf{p}) \sum_{l,m} c_l^m Y_l^m(N_p) \tag{2.17}$$

2.4.2 Computing Spherical Harmonics Coefficients from 2D Photographs

In Eq. 2.17, we are computing total reflected radiance from a point in 3D for diffuse surfaces illuminated by a distant light source. In digital photography, the light reflected from a 3D scene passes through the camera lens, forming a perspective projection onto the 2D sensor. This projection undergoes a couple of filters before converting into pixel values. If we assume

a linear camera response, we can transform the Eq. 2.17 for pixel value $\mathbf{i}(x, y)$, diffuse albedo at that pixel $\rho(x, y)$ and surface normal $N(x, y)$ [60]:

$$\mathbf{i}(x, y) = \rho(x, y) \sum_{l,m} c_l^m Y_l^m(N(x, y)) \tag{2.18}$$

If we repeat this process for all pixels in the image, we can transform the Eq. 2.18 into the following matrices representation, where, for N pixels, vector $\mathbf{i} \in \mathbb{R}^N$ contains pixel values, vector $\rho \in \mathbb{R}^N$ contains diffuse albedo values, matrix $\mathbf{M} \in \mathbb{R}^{N\times 9}$ contains spherical harmonics basis function values for each pixel, vector $\mathbf{l} \in \mathbb{R}^9$ contains the spherical harmonics coefficients, $\otimes$ indicates the Kronecker product and $\odot$ is Hadamard product:

$$\mathbf{i} = (\rho \otimes \mathbf{1}_{(1\times 9)}) \odot \mathbf{Ml} \tag{2.19}$$

We can solve the Eq. 2.19 for $\mathbf{l}$ using least squares approximation as described in equation:

$$\mathbf{l} = ((\rho \otimes \mathbf{1}_{(1\times 9)}) \odot \mathbf{M})^{-1}\mathbf{i} \tag{2.20}$$

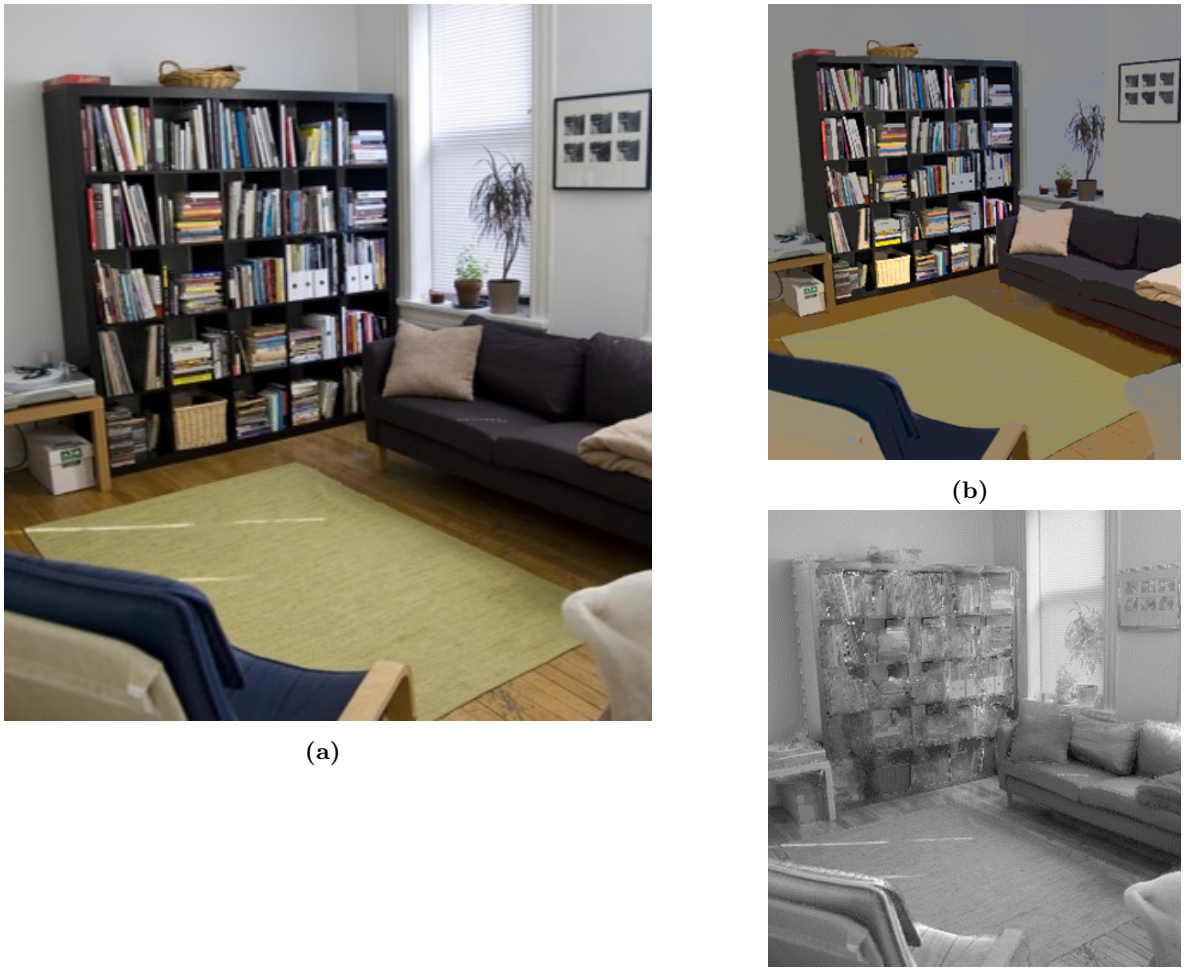
We will use this formulation to compute spherical harmonics coefficients.

2.5 Inverse Rendering

As it is clear from the equation 2.18, we need diffuse albedo and surface normal at each pixel to compute the spherical harmonics coefficients. Johnson *et al.*[60] assume the constant reflectance and skip the diffuse albedo computation. For computing surface normal, they apply a heuristic: take surface points along the occluding boundary as the z-component of the surface normal (N_{p_z}) will be zero for these points [60]. We skip these assumptions and apply IR to compute the diffuse albedo and surface normal.
IR is closely related to Intrinsic Image Decomposition (IID) which is a quite old problem in computer vision [70]. In IID, we try to recover the properties of the 3D scene *i.e.* reflectance, surface normal, and illumination from 2D images. It is an ill-posed problem as we are going from 2D to 3D. And for a given image, there could be multiple possible explanations [70]. A photograph could be an image of the real scene, or it could be an image of another photograph. To solve IID, we make assumptions (*i.e.* smooth illumination variation) about the 3D scene [70] which can help us simplify the solution space. A basic formulation for IID is inspired by human visual system [71] where we decompose an image into reflectance and shading terms as described in Eq 2.21 where R is reflectance, and S is shading. The reflectance captures color information, and shading captures the variations in illumination. An exemplary such decomposition is given in the Figure 2.20.

$$\mathbf{I} = R \times S \tag{2.21}$$

IR tackles a similar problem of learning intrinsic scene properties but is generally studied in the computer graphics community as a tool to relight a rendered scene, insert objects into the scene, and synthesize novel views. As the name suggests, IR is the opposite process of



(a) **(b)** **(c)**

Figure 2.20: An exemplary image decomposition into reflectance and shading taken from Bell *et al.* [68]. (a) original image, (b) estimated reflectance, (c) estimated shading

forward rendering. In forward rendering, we have the 3D information of the objects in the scene, their associated BRDF, and the lighting information, and we try to render the scene using rasterization or ray-tracing. In IR, we have a rendered scene and try to recover the BRDFs, lighting, and scene's 3D geometry. Like IID, IR is also inherently ill-posed, and we make assumptions about the scene to derive a solution. Many traditional methods require multiple images of the scene to constrain the solution space as IR from a single image is a more complex problem [72].

With the availability of ground truth datasets [73, 74, 75, 76, 77], single image IR methods have gained momentum. Single image IR methods can be divided into two categories. The first is optimization-based methods that use strong statistical priors for illumination, reflectance, and shape [72, 73]. The second one uses DNNs to directly regress the reflectance, shape, and illumination in a data-driven manner [78, 18, 20, 79, 19]. We use the method proposed by Yu *et al.* [18]. It is a DNNs based architecture where a shared encoder is used to learn a latent

representation, and three different decoders are used to directly regress the diffuse albedo, surface normal, and cast shadow. We have selected this method as it also uses spherical harmonics coefficients to model illumination. Our inverse rendering pipeline is described in section 4.1 in more detail.

Chapter 3

Dataset Creation

Training data plays a pivotal role in all DL projects. The quality of data directly influences the overall performance of the trained model. There are many datasets of both real and generated images, but ensuring that they contain samples from a similar distribution is challenging. For instance, if real images primarily feature outdoor scenes while generated images mostly contain indoor scenes, the DL network might learn the discrepancies between indoor and outdoor lighting rather than identifying lighting inconsistencies between real and generated images in general. As a result, it may not generalize well. To make sure both real and generated images are samples from a similar distribution, we create a new dataset of real and generated images by conditioning them on the same captions. For that, we get images of in-the-wild real scenes, create captions of these scenes, and use them to create generated images. The following section discusses more details about the data creation pipeline and different statistics of the created dataset.

3.1 Construction of the Training Dataset

For the dataset creation pipeline, we manually collect a few reference images of different real scenes from the internet. These reference images are from uncontrolled environments that typically reflect real-world conditions, including diverse lighting, backgrounds, subjects, and situations. Figure 3.1 shows some of these reference images. We sample images similar to the reference images from LAION-2B-en [80] using a tool called Clip-Retrieval [81]. The LAION-2B-en dataset contains 2.32 billion English image-text pairs mostly scraped from the internet. The Clip-Retrieval tool can perform a k-nearest-neighbor search over the vector database containing clip embedding of all LAION-2B-en dataset. The vector database containing clip embedding for LAION-2B-en was also provided by the authors of the dataset. We sample nearly 7000 images from LAION-2B-en using Clip-Retrieval. We remove duplicate images using perceptual hashing [82]. We also removed images with white backgrounds as those images were providing incorrect lighting estimates. It is important to note that some of the images in our dataset may not be entirely *real*, as they are ultimately collected from the internet.

We create captions of these real images using GIT [83]. GIT is a transformer-based archi-

(a)

(c)

(d)

(b)

(e)

Figure 3.1: Some of the reference images that were used to build the collection of real images by sampling from LAION-2B-en dataset using k-nearest-neighbor search with clip embedding

tecture that uses an image encoder and a text decoder in a unified Vision Language Model (VLM). Additional details about the GIT architecture can be found in its paper [83]. We used pre-trained weights and implementation provided by HuggingFace [84]. We use these captions to generate a collection of generated images using SD v1.5 [53]. The reason for selecting SD is the open-source availability of its architecture and pre-trained weights. We could deploy it locally and create as many images as we want with more control over different network hyperparameters. We used the implementation and pre-trained weights provided by Hugging-

Face [85]. We used PNDM [86] for the sampling process during the reverse diffusion process. PNDM can generate higher quality synthetic images with only 50 steps [86]. We used the implementation provided by HuggingFace diffusers [87] library. We used the guidance scale of 7.5 (from classifier-free guidance [88]) with a batch size of 8. Image dimensions were kept at 512×512. All images are JPEG compressed. Figure 3.2 depicts the data creation pipeline. A few samples containing real images, generated captions, and generated images are shown in Figure 3.3.

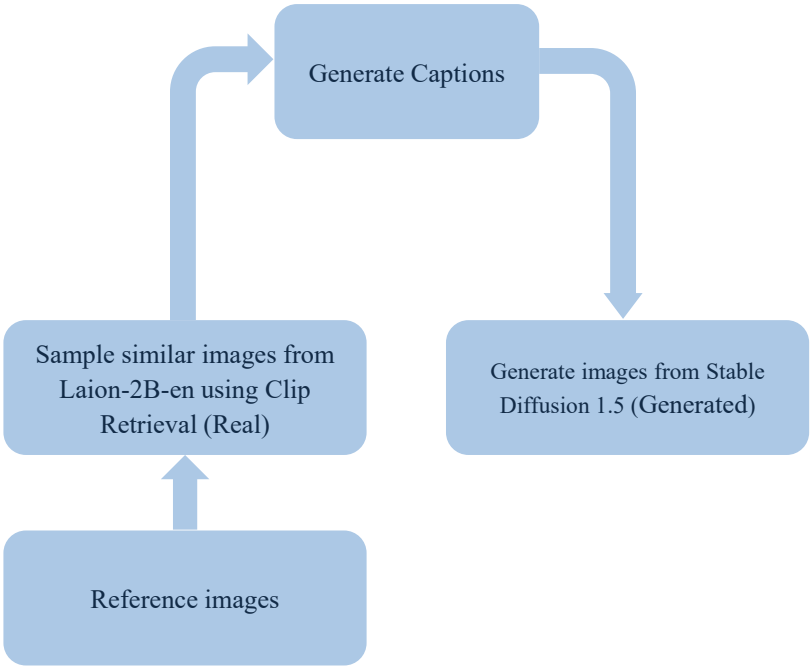


Figure 3.2: The dataset creation pipeline

Our dataset contains 6439 real images and 6899 generated images. The number of real images is slightly less than the generated images, as some real images were removed because their background was white. We created training, validation, and test splits from this dataset using random sampling to train binary classifiers. Table 3.1 shows the distribution of these splits. We use the training split to train different binary classifiers discussed in Chapter 4. Within the context of this report, the term *test set* refers to the test set from this dataset.

Table 3.1: Distribution of real and generated images across data splits

Split	Real Images	Generated Images	Total Images
Train	5123	5501	10624
Validation	665	679	1344
Test	651	719	1370

3.2 Construction of the Cross-generator Evaluation Dataset

To evaluate the generalization ability of our trained binary classifiers on images generated from different generators, we use the Synthbuster [17] dataset. This dataset comprises a total of 9,000 images sourced from nine different generators, with 1000 images from each generator. These nine generators are the following:

1. Glide [89]: It is DDPM-based text-conditioned image generator released by OpanAI [90]. It can generate new images or perform inpainting on existing images using text prompts. For text-conditioning, it trains a transformer [43] on image captions to get text embedding. These text embedding are then used for conditioning during the diffusion process. The original architecture outputs 64×64 resolution, which then are upsampled to 256×256 using another upsampling diffusion model [89].

2. DALL-E2 [91]: It is a commercial text-conditioned image generator by OpenAI. The exact implementation details might differ, but we can get insights from the associated research paper [91], where it is referred to as unCLIP. To generate text-conditioned images, text embedding is computed from text prompts using a CLIP text encoder. These text embedding are transformed to image embedding using a transformer. These image embedding are then used for conditioning in DDPM-based image generator [91]. The generated images are of 64×64 resolution, which are then upsampled twice by two different diffusion models to generate images of 1024×1024 resolution.

3. DALL-E3 [92]: It is a successor of DALL-E2, released by OpenAI in 2023. Exact implementation details might differ as it is a commercial text-conditioned image generator. According to the associated research paper [92], their original training set contains noise and incorrect image captions. They train an image captioner first and use the generated captions to train a text-conditioned image generator. The diffusion process is performed directly in image space for DALL-E2; it is speculated that the diffusion process is performed in latent space for DALL-E3 in LDM-style [53].

4. Adobe Firefly [3]: It is a text-conditioned image generator released by Adobe. The specific details of the architecture remain undisclosed, although some information about the training dataset has been shared. The training dataset contains Adobe Stock images, openly licensed content, and public domain content.

5. Midjourney v5 [93]: It is version 5 of the text-conditioned image generator, released by Midjourney [1]. There is no publicly available information on either the dataset or the architecture.

6. Stable Diffusion v1.3 [94]: It is LDM-based text-conditioned image generator, released by computer vision and learning group at LMU Munich. It is trained on LAION-5B [80] dataset. For text conditioning, it uses a pre-trained CLIP text encoder to get text embedding. Initially, it is pre-trained on LAION-2B-en, a subset of LAION-5B, at a resolution of 256×256. It is then fine-tuned on LAION-high-resolution and LAION-Aesthetics V2.5+, subsets of LAION-5B, at 512×512 resolution [94].

7. Stable Diffusion v1.4 [94]: It is similar to Stable Diffusion v1.3 except it is fine-tuned on LAION-Aesthetics V2.5+ for more training steps than Stable Diffusion v1.3.

8. Stable Diffusion v2 [95]: It is released by Stability AI. Its training process is similar to Stable Diffusion v1.3, except it uses OpenCLIP [96] as a text encoder to get text embeddings. In addition to being pre-trained at a resolution of 256×256 and fine-tuned at 512×512, it is also further fine-tuned at a resolution of 768×768.

9. Stable Diffusion XL [97]: It is based on ensemble of diffusion models [98]. The ensemble consists of two diffusion models. The first diffusion model generates the image, while the second diffusion model adds high-quality details.

The creation of the Synthbuster dataset follows the pipeline similar to the one discussed in the previous section for creating the training dataset. They use 1000 real images from the RAISE-1K [99] dataset. They create captions for these images and use these captions to create images from different generators. We extend the dataset by adding images from Stable Diffusion v1.5. We get the captions of RAISE-1K images from the Synthbuster dataset and use them to create images from Stable Diffusion v1.5. We skipped the Glide images from the evaluation dataset as the images were not photo-realistic compared to other generators. We also perform JPEG compression with 90% quality on all images, as the original Synthbuster dataset contains PNG images. We report the results on this dataset in Chapter 5. We never used this dataset to train any of our binary classifiers.

A margarita with a lime wedge and a straw

A man standing next to a statue of a man

A bust of a woman, made of plaster

A castle on a hill overlooking a river

Figure 3.3: Samples from our training dataset. Real images (on the left), generated captions (in the center), and generated images from SD v1.5 (on the right)

(a)

(b)

(c)

(d)

(e)

Figure 3.4: Samples from cross-generators evaluation dataset (a) A real image from RAISE-1K dataset (b) A corresponding image from Synthbuster dataset generated from the SD v2.0 (c) Generated image from Adobe Firefly (d) Generated image from Midjourney v1.5 (e) Generated image from DALL-E3

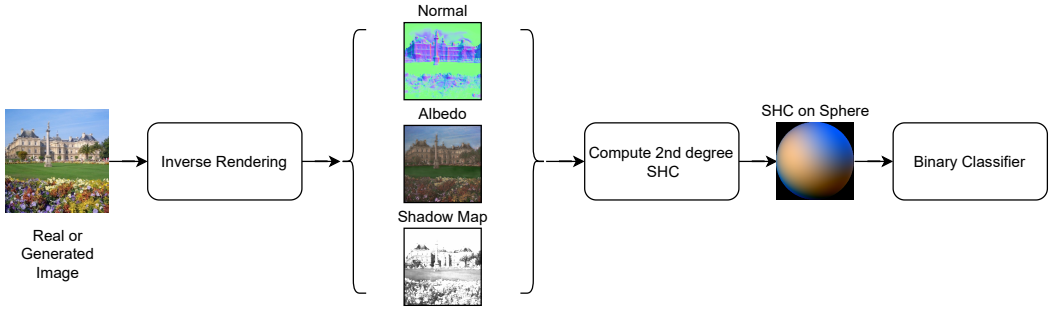
Chapter 4

Methodology

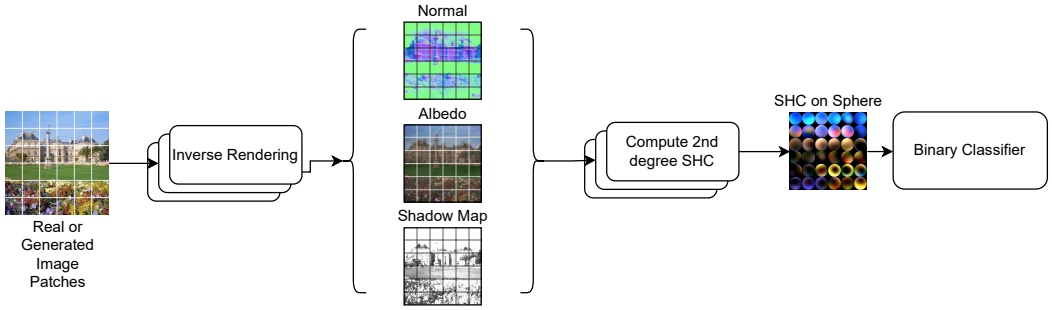
Our method is based on computing second-degree Spherical Harmonics Coefficients (SHC) for both real and generated images and using them as an approximation for scene lighting conditions. To distinguish between real and generated images, we train various binary classifiers using SHC values as input. For second-degree spherical harmonics, there are 9 coefficients, and we estimate these 9 coefficients individually for each RGB color channel resulting in 27 coefficients.
We explore two different strategies which we call SHL(g) and SHL(l). In SHL(g), we process the whole image simultaneously and compute 27 coefficients. In SHL(l), we divide the image into n patches and compute 27 coefficients, resulting in $n \times 27$ parameters. Our proposed pipeline is shown in Figure 4.1. Before presenting the results of our pipeline, we introduce several baseline methods and discuss the results obtained from these baselines. Subsequently, we will present the results of our proposed method and compare them against the baseline results.

4.1 Inverse Rendering Pipeline

The process of computing SHC begins with IR to estimate intrinsic image properties, such as the surface normal map, albedo map, and shadow map. We use the IR pipeline proposed by Yu *et al.* [18], which we refer to as IRN++ within the context of this report. The reason for selecting IRN++ is that it also uses spherical harmonics representation for approximating scene lighting, and its code and trained network weights are available [100]. IRN++ approaches the IR as an image-to-image translation by directly regressing the surface normal, albedo, and shadow maps [18]. It uses an encoder-decoder architecture where a shared encoder is used to get the latent image representation, and a separate decoder is used to estimate each map. Both encoder and decoder are CNNs.
The predicted albedo map contains three channels for RGB output, and the shadow map consists of only a single channel, and its values are in the $[0, 1]$ range. The predicted shadow map is not physically consistent shadow estimation, but it acts as a scaling factor on the albedo estimation [18]. As the surface normal has $||n||_2 = 1$, it has two degrees of freedom despite being a 3D vector. Therefore, the predicted surface normal map consists of two quantities,



(a) SHL(g): Compute Spherical Harmonics Coefficients (SHC) for the whole image



(b) SHL(l): Divide the images into patches, perform inverse rendering on each patch, and compute Spherical Harmonics Coefficients (SHC) for each patch

Figure 4.1: Our proposed method. In (a), we run the pipeline on the whole image, and in (b), we divide the image into patches and compute coefficients for each patch. We then accumulate them and pass them to the binary classifier. The renderings of SHC on the sphere are just for visualization as we pass SHC to the binary classifier directly

n_x/n_z and n_y/n_z. The actual surface normal is then computed using the following equation:

$$\mathbf{n} = \frac{[n_x/n_z, n_y/n_z, 1]^T}{\|[n_x/n_z, n_y/n_z, 1]\|} \tag{4.1}$$

These maps are then used to estimate the SHL. The estimated SHL is then used with the surface normal map and albedo map to render the scene. For supervision during the training, it uses perceptual loss [101] between the rendered scene and shadow-free original scene [18]. IRN++ propose pipeline is shown in Figure 4.2. It can be seen that IRN++ masks out the sky region before feeding the image to the network but we skip this sky masking in our pipeline as it adversely relates to the performance of binary classifiers in our experiments. IRN++ is trained on MegaDepth [102] dataset. During our experiments, we used the pre-trained weights to estimate lighting by skipping the rendering part. The IRN++ also incorporates a natural illumination prior to constraining the space of possible illumination to the natural environments [18]. However, our experimental results were indifferent to the suggested prior. Therefore, we show results without adding the natural illumination prior. An exemplary IRN++ output is shown in Figure 4.3.

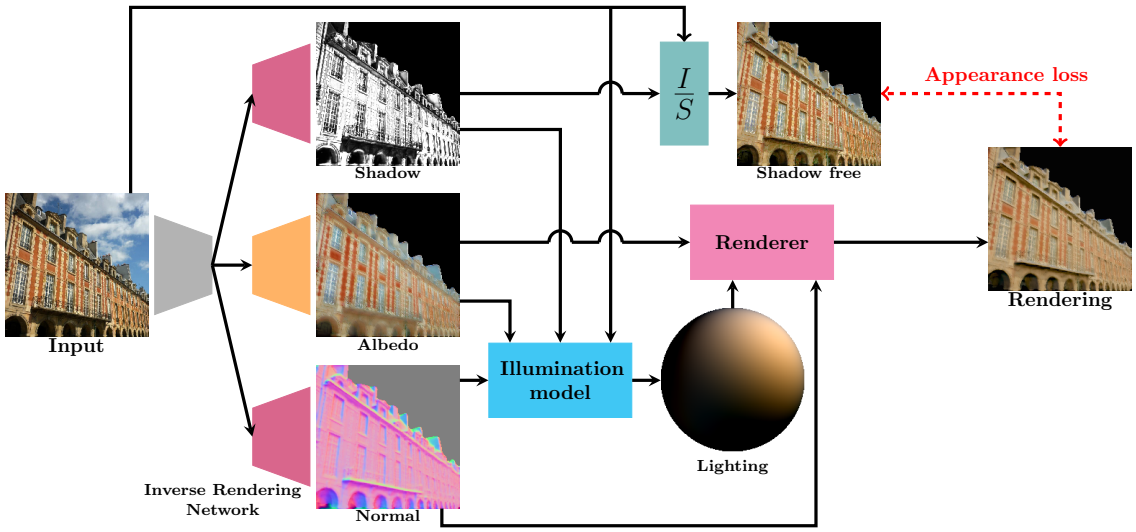


Figure 4.2: IRN++ proposed pipeline [18]. We use estimated lighting from this pipeline and skip the rendering part

4.2 Computation of Spherical Harmonics Coefficients

Ramamoorthi *et al.* [103, 32] is a good reference for spherical harmonics representation of scene lighting. We use the IRN++ pipeline to compute SHC. For clarity and reference to the possible derived work, we take the derivation process from Yu *et al.* [18] and document it here.

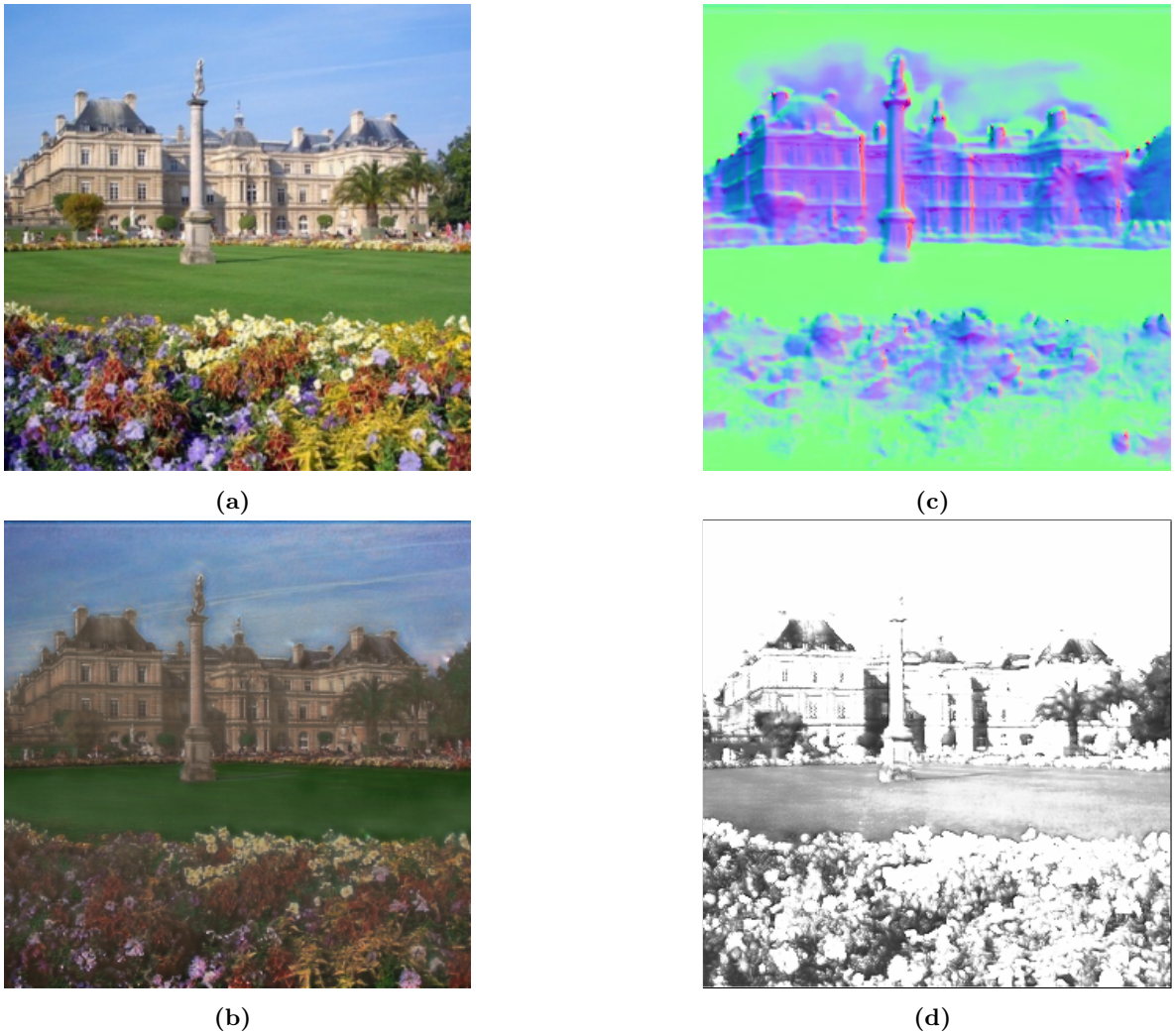
To compute SHC for Lambertian surfaces illuminated by a distant illumination source, we rewrite the Eq. 2.18:

$$\mathbf{i}(x,y) = \rho(x,y)E(n(x,y)) \tag{4.2}$$

Yu *et al.*[18] models $\rho(x,y) = \mathbf{s}(x,y)\alpha(x,y)$; predicted diffuse albeldo α is scaled by a predicted shadow term $\mathbf{s}$. Irradiance E is a function of the surface normal $n = [n_x, n_y, n_z]$ only and is given by the polynomial where L_{lm} denotes the spherical harmonics coefficient up to the order 2 *i.e.* $l \leq 2$ and $-l \leq m \leq l$ and the constants are defined as $c_1 = 0.429043, c_2 = 0.511664, c_3 = 0.743125, c_4 = 0.886227, c_5 = 0.247708$ [32]:

$$\begin{aligned} E(\mathbf{n}) = {} & c_1 L_{22}(n_x^2 - n_y^2) + c_3 L_{20} n_z^2 + c_4 L_{00} - c_5 L_{20} \\ & + 2c_1(L_{2-2} n_x n_y + L_{21} n_x n_z + L_{2-1} n_y n_z) \\ & + 2c_2(L_{11} n_x + L_{1-1} n_y + L_{10} n_z) \end{aligned} \tag{4.3}$$

We transform the Eq. 4.3 into a vector product $\mathbf{E} = \mathbf{b^T l}$ where $\mathbf{b} \in \mathbb{R}^9$ and $\mathbf{l} \in \mathbb{R}^9$ are the following:



(a) **(c)** **(b)** **(d)**

Figure 4.3: An exemplary IRN++ IR results without sky masking (a) Input Image, (b) Predicted diffuse albedo, (c) Predicted surface normals (d) Predicted shadow term

$$\mathbf{b} = [c_4, 2c_2n_y, 2c_2n_z, 2c_2n_x, 2c_1n_xn_y, 2c_1n_yn_z, c_3{n_z}^2 - c_5, 2c_1n_zn_x, c_1({n_x}^2 - {n_y}^2)] \tag{4.4}$$

$$\mathbf{l} = [L_{00}, L_{1-1}, L_{10}, L_{11}, L_{2-2}, L_{2-1}, L_{20}, L_{21}, L_{22}] \tag{4.5}$$

We reuse the Eq. 2.19 for N pixels in the following form where $\mathbf{M} = [b(n_1), b(n_2), ...b(n_N)]$, $\otimes$ indicates the Kronecker product, and $\odot$ is Hadamard product:

$$\mathbf{i} = (\rho \otimes \mathbf{1}_{(1\times 9)}) \odot \mathbf{Ml} \tag{4.6}$$

We can solve the Eq. 4.6 for $\mathbf{l}$ using least squares approximation:

$$\mathbf{l} = ((\rho \otimes \mathbf{1}_{(1\times 9)}) \odot \mathbf{M})^{-1}\mathbf{i} \tag{4.7}$$

We repeat the above process with red, green, and blue channels separately to get 27 SHC.

4.3 Binary Classification

4.3.1 Evaluation Metrics

To evaluate the performance of different binary classifiers, we use the balanced classification accuracy metric. We use the following expressions to compute balanced classification accuracy, where R and G represent the total real and generated images in the evaluation dataset, and R' and G' represent the images correctly predicted real and generated, respectively:

$$\text{Real Images Accuracy } (R_{Acc}) = \frac{R'}{R}$$

$$\text{Generated Images Accuracy } (G_{Acc}) = \frac{G'}{G}$$

$$\text{Balanced Accuracy} = \frac{1}{2}\left(R_{Acc} + G_{Acc}\right)$$

4.3.2 Binary Classifiers

To explore different linear and non-linear decision boundaries between real and generated samples, we experiment with different binary classifiers. We tried six different binary classifiers, which include Random Forest (RF), Support Vector Machine (SVM), Logistic Regression (LR), Multi-Layer Perceptron (MLP), and Vision Transformer (ViT). For SVM, we experiment with linear kernel SVM(L) and radial basis function kernel SVM(RBF). For RF, we set the max depth to 5 as it performed better in our experiments. For RF, SVM, LR, we use the scikit-learn [104] implementation with default parameters.
Our MLP classifier consists of three hidden linear layers where each layer is followed by a ReLU activation function and a dropout layer [36], which randomly drops 50% of the features. This helps to reduce overfitting [36]. Our MLP architecture is shown in Figure 4.4. To train the MLP, we use the following parameters:

- Optimizer: AdamW [105]
- Initial learning rate: 0.0005
- Weight decay: 0.05
- Learning rate scheduler: ReduceLROnPlateau (factor=0.1, patience=2) [106]
- Early stopping: Yes (patience=4)

For ViT experiments, we use ViT proposed by Dosovitskiy *et al.* [44] with patch size 32 and image size 224x224. We use the implementation by HuggingFace [107]. We only use ViT for SHL(l) experiments where we have 27 spherical harmonics coefficients for each patch. We re-implement the initial patch embedding layer to incorporate the spherical harmonics coefficients in patch embedding. We experiment with the following three scenarios:

- ViT(Imgs): That is similar to traditional ViT with image patches.

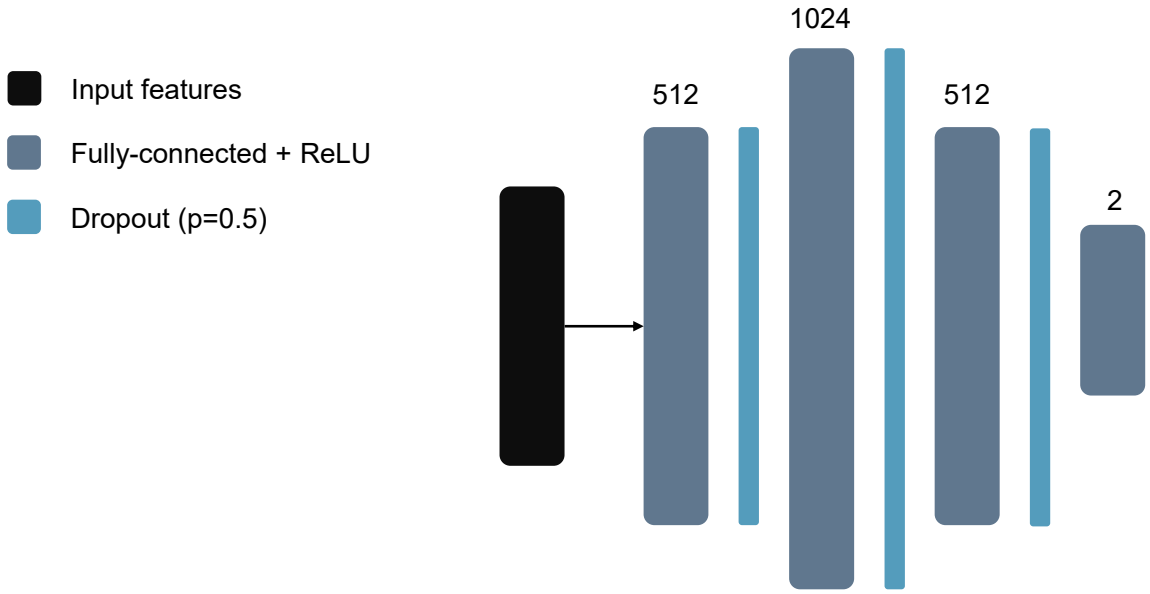


Figure 4.4: The MLP architecture used for binary classification

- ViT(SHL(l)): Here we use 27 spherical harmonics coefficients padded by zeros instead of image patch embedding.
- ViT(Imgs+SHL(l)): Here we concatenate image patch embedding and spherical harmonics coefficients for each patch.

We also changed the last classification layer, as our output classes are two. We experiment with pre-trained weights as well as weights reinitialized from scratch again. For training, we use the Adam [34] optimizer with an initial learning rate of 0.0001. We trained our network for 10 epochs by visualizing the validation loss.

4.3.3 Data Preprocessing

For all our experiments except ViT ones, we resize the image's lowest dimension to 300 while the other dimension is computed according to the aspect ratio. We then take a center crop of 300x300. In SHL(l) experiments, we use a patch size of 50.
In the case of ViT experiments, we follow the same process, but our final image dimensions are 224x224, and the patch size is 32.

4.3.4 Baselines

Before showing results for our proposed pipeline, we explore different baselines that we can use to compare our results. We select three different architectures that are based on totally different ideas. We select pre-trained CLIP [46] as it has been used in many state-of-the-art detectors recently [7, 21] where they show that CLIP with simple linear classifiers outperforms other DNNs classifiers on unseen images of newer AI-image generators. CLIP is trained on

a large amount of internet collected images in a self-supervised manner. It is possible that it learns the distribution of real images and can be used to discriminate from generated images. To study the differences between real and generated images in the frequency domain, we use GIST [108] to exploit the frequency space differences as different AI-image generators have different artifacts in frequency space [17]. Recently, DNNs-based compression networks *e.g.* HiFiC [109] are proposed for image compression where they project input image into a compact latent representation and then use a decoder to reconstruct the image from latent representation. We use HiFiC as a baseline to exploit the differences in compact latent representation.

4.3.4.1 CLIP

We have discussed the architectural details of CLIP in section 2.1.0.5. Here we will discuss the implementation details for our use case. To classify images into real and generated, we only use the CLIP image embedding. We use the CLIP implementation and pre-trained weights from OpenAI [110]. For image encoder, it has multiple implementations of ViT with different patch sizes. We have used the ViT-B with patch size 32. We pass every image to the CLIP image encoder and get 512-dimensional output embedding. These embeddings are then used to train binary classifiers.

4.3.4.2 GIST

GIST is a computational model that can be used for the classification of real-world scenes [108]. GIST works by evaluating scenes across five perceptual dimensions: naturalness, openness, roughness, expansion, and ruggedness [108]. The degree of naturalness indicates the differences between man-made and natural scenes. Man-made scenes, i.e. buildings, tend to have more horizontal and vertical lines than natural scenes, while natural scenes have textured zones and undulating contours [108]. A scene having properties of a natural scene would have a higher degree of naturalness. The degree of openness shows whether a scene is vast or contains some spatial enclosure with visual references, i.e. mountains, forest, etc. The existence of the horizon line correlates with a better degree of openness. The degree of roughness relates to the size of the major elements in the scene and their ability to make complex environments. The degree of expansion further divides the man-made scenes based on the perception of the depth. A street view with long vanishing lines has a higher degree of expansion than a tall building [108]. The degree of ruggedness refers to the deviation of the ground with respect to the horizon [108]. Mountains have a higher degree of ruggedness than open scenes.
To classify scenes based on the perceptual scene properties, a *Spatial Envelop* is estimated using spectral and coarsely localized information [108]. Concretely speaking, an image is divided into a 4×4 grid, and a bank of Gabor filters is applied on each patch. The filter responses for all patches are accumulated to make a single spectral signature of a scene. This spectral signature can be used to classify different real-world scenes.
Because GIST creates spectral signatures based on the statistics of real-world scenes, we try to explore its usage for discriminating AI-generated images with the assumption that the

spectral signature of AI-generated images will differ from images of real scenes.
We use the original GIST implementation [111] but rewrite the code from MATLAB to Pytorch. The number of Gabor filters used for spectral signature is 32. As there are 16 patches, we get 512-dimensional spectral signatures for each image. We use this signature to train our binary classifiers.

4.3.4.3 HiFiC

High-Fidelity Generative Image Compression (HiFiC) [109] is a DNN-based lossy image compression network. It reconstructs perceptually similar output at an extremely low bit rate by utilizing a conditional GAN for reconstruction. The HiFiC architecture is shown in Figure 4.5, where an image x is first converted into a latent representation y using an encoder E. The latent representation is then passed through a hyper-prior probability model P, where it is further compressed to hyper-latent representation z and then upscaled again to model $p(y|z)$. The generator G of conditional GAN receives $p(y|z)$ and upscale it to estimate reconstructed output x'. The discriminator receives y and x', concatenates them, and predicts if samples are real or reconstructed. For training, perceptual loss [101] between real and reconstructed images is used. A collection of images collected from the internet is used for training.
We get the HiFiC pre-trained weights and Pytorch implementation from this GitHub repository [112]. As we are only interested in compressed representation for classifying real and generated images, we do not use conditional GAN. We use the hyper-latent representation z without the entropy encoding model (represented as $\diamond$ in Figure 4.5) and flatten it to make an 8000-dimensional vector. We use this vector to train our binary classifiers.

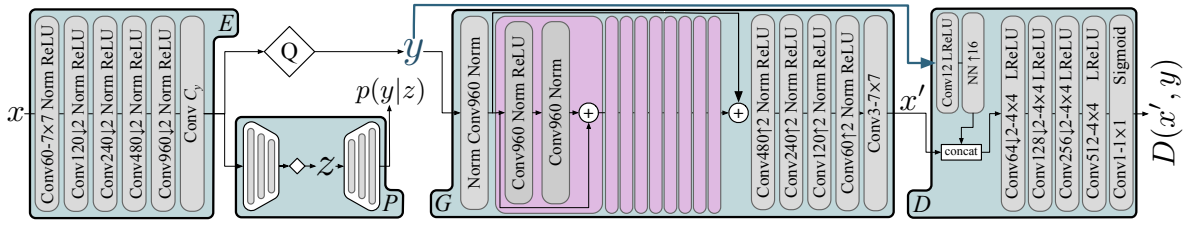


Figure 4.5: The HiFiC architecture, proposed by [109]. We use the hyper-latent z without entropy encoding model (represented as $\diamond$) for classifying images into real or generated.

Chapter 5

Results

We present results for baseline methods and our proposed method on test set and cross-generator evaluation dataset. We report results using balanced classification accuracy. We chose balanced classification accuracy because our cross-generator evaluation dataset is imbalanced, consisting of 1000 real and 9000 generated images.
This chapter is organized as follows: we report results for CLIP, GIST, and HiFiC in section 5.1.1 as baseline, results for our proposed methods, SHL(g) and SHL(l) with different binary classifiers excluding ViT are reported in 5.2, in 5.3 we report results by concatenating features that we get from baseline methods, and SHC from SHL(g) and SHL(l). The best results for our proposed method are obtained using ViT, which are reported in 5.4. In the last section 5.5, we compare our best results to the best results obtained from baseline methods.

5.1 Baseline Results

5.1.1 Evaluation on Test Set

Table 5.1 presents results on the test set for CLIP, GIST, and HiFiC. CLIP outperforms GIST and HiFiC by getting 99% accuracy obtained with logistic regression. CLIP also gets high performance with MLP and SVM classifiers, but its performance drops with RF. As CLIP is trained on a large amount of real images with captions, it is possible that it has a semantic understanding of real scenes. However, we cannot explicitly tell what features CLIP uses to discriminate real and generated images. HiFiC also appears to have strong discriminatory features indicating possible differences in compressed latent representation for real and generated images. GIST features do not have strong classification performance, indicating that generated and real images have similar spectral signatures. The Gabor filters used to compute GIST spectral signature were initially designed for outdoor scenes [108]; it is possible that these filters are not suited for in-the-wild indoor and outdoor scenes.

5.1.2 Evaluation on Cross-generator Dataset

CLIP, GIST, and HiFiC performance on cross-generator evaluation dataset is presented in Table 5.2, 5.3 and 5.4 respectively. CLIP obtains the best performance of 76% using MLP.

Type	Features	RF	SVM (RBF)	SVM (L)	LR	MLP
Test Set	CLIP	0.89	0.95	0.94	**0.99**	0.97
	GIST	0.67	**0.77**	0.68	0.72	0.71
	HiFiC	0.78	**0.86**	0.81	0.82	0.85

Table 5.1: Balanced classification accuracy of baseline methods on test set with different binary classifiers. Best performing results are marked in bold.

CLIP performance is high on SD v1.3 and SD v1.4, which generate images similar to SD v1.5, the generator used for creating a training set. CLIP's high performance on DALL-E3 indicates the likelihood of the diffusion process in DALL-E3 being done in latent space similar to latent diffusion models. CLIP performance is poor on Firefly, DALL-E2, and Midjourney, indicating an inability to detect generated images when there are substantial architectural changes in the generator. The architectural details for Firefly and Midjourney are not available, but it can be speculated that they generate images in a manner different from that of latent diffusion models. The HiFiC and GIST perform better than random guess but worse than CLIP on cross-generator data, indicating that learned features are not transferable to other generators. The HiFiC and GIST also follow a similar trend: a high performance on SD v1.3 and SD v1.4 and low performance on Firefly, DALL-E2, and Midjourney.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.82	0.95	0.96	0.97	0.96
Generated	Adobe Firefly	0.31	0.14	0.13	0.13	0.17
	SD v1.3	0.84	0.88	0.86	0.95	0.96
	SD v1.4	0.86	0.88	0.87	0.96	0.97
	SD v1.5	0.90	0.94	0.94	0.98	0.98
	SD v2.0	0.75	0.46	0.41	0.30	0.35
	SD XL	0.64	0.46	0.42	0.33	0.42
	DALL-E2	0.42	0.18	0.15	0.15	0.18
	DALL-E3	0.80	0.82	0.83	0.66	0.79
	Midjourney v5	0.46	0.25	0.22	0.17	0.23
Real Images Accuracy		0.82	0.95	0.96	**0.97**	0.96
Generated Images Accuracy		**0.66**	0.55	0.53	0.52	0.56
Balanced Classification Accuracy		0.74	0.75	0.75	0.74	**0.76**

Table 5.2: CLIP performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.46	0.84	0.76	0.78	0.93
Generated	Adobe Firefly	0.54	0.21	0.31	0.28	0.12
	SD v1.3	0.86	0.66	0.76	0.77	0.54
	SD v1.4	0.86	0.63	0.75	0.76	0.51
	SD v1.5	0.85	0.72	0.68	0.69	0.55
	SD v2.0	0.94	0.27	0.42	0.34	0.16
	SD XL	0.77	0.36	0.50	0.45	0.27
	DALL-E2	0.65	0.38	0.41	0.37	0.23
	DALL-E3	0.81	0.61	0.57	0.51	0.38
	Midjourney v5	0.61	0.35	0.43	0.40	0.22
Real Images Accuracy		0.46	0.84	0.76	0.78	**0.93**
Generated Images Accuracy		**0.73**	0.44	0.51	0.48	0.31
Balanced Classification Accuracy		0.59	0.64	0.64	**0.64**	0.62

Table 5.3: GIST performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.71	0.83	0.82	0.87	0.82
Generated	Adobe Firefly	0.34	0.30	0.30	0.25	0.39
	SD v1.3	0.71	0.73	0.62	0.59	0.72
	SD v1.4	0.73	0.73	0.62	0.58	0.70
	SD v1.5	0.82	0.80	0.73	0.73	0.84
	SD v2.0	0.55	0.45	0.50	0.50	0.65
	SD XL	0.39	0.38	0.38	0.36	0.48
	DALL-E2	0.45	0.35	0.35	0.31	0.33
	DALL-E3	0.42	0.38	0.40	0.31	0.42
	Midjourney v5	0.44	0.29	0.35	0.28	0.27
Real Images Accuracy		0.71	0.83	0.82	**0.87**	0.82
Generated Images Accuracy		0.53	0.46	0.45	0.42	0.50
Balanced Classification Accuracy		0.62	0.65	0.63	0.65	**0.66**

Table 5.4: HiFiC performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

5.2 Using Spherical Harmonics Coefficients

5.2.1 Evaluation on Test Set

The test set results for our proposed SHL(g) and SHL(l) methods are presented in Table 5.5. SHL(l) performs better and gets 77% accuracy with MLP. The lower performance for SHL(g) indicates that irradiance environment maps computed for a whole image in generated images are somewhat similar to real images, but there are inconsistencies if we compute irradiance environment maps in a patch-wise manner. These results are worse than what we get from baseline methods, indicating that either our simple binary classifiers are not able to exploit the lighting inconsistencies or that not all generated images have lighting inconsistencies.

Type	**Features**	**RF**	**SVM (RBF)**	**SVM (L)**	**LR**	**MLP**
Test Set	SHL(g)	0.65	0.68	0.65	0.68	**0.70**
	SHL(l)	0.69	0.73	0.71	0.73	**0.77**

Table 5.5: Balanced classification accuracy for SHL methods on test set with different binary classifiers. Best performing results are marked in bold.

5.2.2 Evaluation on Cross-generator Dataset

The results of the cross-generator evaluation dataset for SHL(g) and SHL(l) are presented in Table 5.6 and 5.7, respectively. SHL(g) and SHL(l) perform similarly on the cross-generator, getting 61% and 62% balanced accuracy. The results are worse than what we get from baseline methods but still better than random guess. Although, the decision boundary differs from baseline methods *e.g.* CLIP. With CLIP features, classifiers consider many generated images to be real. Here, the classifier has better performance on generated images, but many real images are considered as generated. Another important difference between SHL methods and CLIP is that they have less tendency to overfit similar generator images. CLIP performance is high on SD v1.3 and SD v1.4 but drops significantly on Firefly, DALL-E2, and Midjourney. With SHL methods, we have better performance on Firefly, DALL-E2, and Midjourney, but overall performance is low because of low performance on real images.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.22	0.50	0.57	0.50	0.49
Generated	Adobe Firefly	0.76	0.50	0.45	0.53	0.45
	SD v1.3	0.90	0.64	0.75	0.84	0.73
	SD v1.4	0.92	0.68	0.75	0.86	0.72
	SD v1.5	0.85	0.68	0.67	0.77	0.76
	SD v2.0	0.96	0.75	0.86	0.91	0.74
	SD XL	0.94	0.72	0.79	0.85	0.71
	DALL-E2	0.61	0.54	0.39	0.47	0.50
	DALL-E3	0.85	0.67	0.67	0.77	0.60
	Midjourney v5	0.77	0.55	0.54	0.62	0.54
Real Images Accuracy		0.22	0.50	**0.57**	0.50	0.49
Generated Images Accuracy		**0.82**	0.62	0.62	0.71	0.63
Balanced Classification Accuracy		0.52	0.56	0.59	**0.61**	0.56

Table 5.6: SHL(g) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.38	0.50	0.56	0.56	0.45
Generated	Adobe Firefly	0.61	0.47	0.53	0.51	0.65
	SD v1.3	0.91	0.79	0.79	0.86	0.90
	SD v1.4	0.91	0.79	0.80	0.86	0.90
	SD v1.5	0.86	0.79	0.77	0.82	0.90
	SD v2.0	0.94	0.80	0.83	0.89	0.81
	SD XL	0.86	0.76	0.66	0.74	0.72
	DALL-E2	0.57	0.49	0.46	0.48	0.59
	DALL-E3	0.79	0.69	0.69	0.70	0.78
	Midjourney v5	0.65	0.56	0.56	0.55	0.62
Real Images Accuracy		0.38	0.50	**0.56**	**0.56**	0.45
Generated Images Accuracy		**0.78**	0.65	0.65	0.68	0.74
Balanced Classification Accuracy		0.57	0.58	0.60	**0.62**	0.60

Table 5.7: SHL(l) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

5.3 Combining Spherical Harmonics Coefficients with Baseline

We combine features from baseline methods and SHC from SHL(g) and SHL(l) by concatenating them with the hope of improving baseline methods' performance on the cross-generator dataset. Another reason for combining features is that, as the baseline methods represent image content in some latent representation, our SHL methods might also improve performance as image content and lighting information will be available for classifiers to find inconsistencies. However, performance on both the test set and cross-generator evaluation set remains somewhat similar to the one we get from baseline methods alone, indicating that our classifiers only focus on baseline features as they have a better discriminatory nature.

5.3.1 Evaluation on Test Set

Table 5.8 presents the results of combined features on the test set. The results are similar to the ones we get from baseline features alone, indicating that binary classifiers are unable to use extra information for better classification.

Type	Features	RF	SVM (RBF)	SVM (L)	LR	MLP
Test Set	CLIP+SHL(g)	0.89	0.95	0.95	**0.98**	0.97
	CLIP+SHL(l)	0.89	0.94	0.94	**0.97**	0.94
	GIST+SHL(g)	0.67	0.76	0.72	0.74	**0.77**
	GIST+SHL(l)	0.70	0.74	0.75	0.77	**0.77**
	HiFiC+SHL(g)	0.78	0.85	0.81	0.84	**0.86**
	HiFiC+SHL(l)	0.77	0.85	0.78	0.81	**0.85**

Table 5.8: Balanced classification accuracy for baselines+SHL methods on test set with different binary classifiers. Best performing results are marked in bold.

5.3.2 Evaluation on Cross-generator Dataset

The results of combined features for CLIP and SHL(g) and SHL(l) are presented in Table 5.9 and 5.10, respectively. Similarly, for GIST and HiFiC, results are presented in Table 5.11, 5.12, 5.13, and 5.14 respectively. The results are similar to the ones we get from baseline features alone, indicating that binary classifiers are unable to use extra information for better classification.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.78	0.95	0.96	0.98	0.97
Generated	Adobe Firefly	0.33	0.15	0.12	0.13	0.14
	SD v1.3	0.86	0.88	0.87	0.96	0.91
	SD v1.4	0.86	0.89	0.87	0.96	0.91
	SD v1.5	0.89	0.94	0.93	0.98	0.97
	SD v2.0	0.77	0.48	0.45	0.31	0.31
	SD XL	0.64	0.48	0.46	0.36	0.37
	DALL-E2	0.43	0.18	0.14	0.13	0.14
	DALL-E3	0.80	0.84	0.83	0.67	0.78
	Midjourney v5	0.44	0.27	0.23	0.16	0.16
Real Images Accuracy		0.78	0.95	0.96	**0.98**	0.97
Generated Images Accuracy		**0.66**	0.56	0.54	0.52	0.51
Balanced Classification Accuracy		0.72	**0.76**	0.75	0.75	0.74

Table 5.9: CLIP+SHL(g) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.67	0.95	0.96	0.97	0.93
Generated	Adobe Firefly	0.42	0.21	0.20	0.16	0.24
	SD v1.3	0.89	0.87	0.88	0.95	0.91
	SD v1.4	0.90	0.87	0.89	0.95	0.91
	SD v1.5	0.91	0.93	0.93	0.97	0.95
	SD v2.0	0.88	0.60	0.66	0.42	0.59
	SD XL	0.75	0.51	0.56	0.40	0.49
	DALL-E2	0.47	0.20	0.19	0.16	0.28
	DALL-E3	0.82	0.86	0.86	0.68	0.80
	Midjourney v5	0.52	0.26	0.30	0.17	0.27
Real Images Accuracy		0.67	0.95	0.96	**0.97**	0.93
Generated Images Accuracy		**0.72**	0.56	0.58	0.53	0.58
Balanced Classification Accuracy		0.69	0.76	**0.77**	0.75	0.76

Table 5.10: CLIP+SHL(l) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.46	0.75	0.72	0.74	0.81
Generated	Adobe Firefly	0.51	0.31	0.35	0.33	0.26
	SD v1.3	0.87	0.65	0.81	0.80	0.62
	SD v1.4	0.86	0.68	0.82	0.80	0.61
	SD v1.5	0.85	0.73	0.75	0.73	0.68
	SD v2.0	0.94	0.32	0.58	0.48	0.21
	SD XL	0.78	0.42	0.70	0.62	0.33
	DALL-E2	0.64	0.42	0.40	0.38	0.39
	DALL-E3	0.80	0.47	0.62	0.56	0.35
	Midjourney v5	0.60	0.35	0.50	0.45	0.29
Real Images Accuracy		0.46	**0.75**	0.72	0.74	0.81
Generated Images Accuracy		**0.73**	0.48	0.59	0.55	0.41
Balanced Classification Accuracy		0.60	0.61	0.65	**0.65**	0.61

Table 5.11: GIST+SHL(g) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.43	0.54	0.73	0.76	0.57
Generated	Adobe Firefly	0.58	0.45	0.41	0.36	0.58
	SD v1.3	0.88	0.72	0.74	0.71	0.71
	SD v1.4	0.89	0.73	0.73	0.68	0.71
	SD v1.5	0.86	0.78	0.74	0.73	0.81
	SD v2.0	0.95	0.60	0.41	0.37	0.45
	SD XL	0.82	0.60	0.51	0.47	0.54
	DALL-E2	0.60	0.48	0.41	0.35	0.52
	DALL-E3	0.79	0.63	0.56	0.51	0.64
	Midjourney v5	0.64	0.51	0.38	0.37	0.48
Real Images Accuracy		0.43	0.54	0.73	**0.76**	0.57
Generated Images Accuracy		**0.75**	0.59	0.52	0.48	0.58
Balanced Classification Accuracy		0.59	0.56	0.62	**0.62**	0.58

Table 5.12: GIST+SHL(l) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.68	0.83	0.83	0.89	0.87
Generated	Adobe Firefly	0.35	0.30	0.29	0.24	0.29
	SD v1.3	0.71	0.73	0.64	0.63	0.69
	SD v1.4	0.74	0.75	0.63	0.64	0.68
	SD v1.5	0.84	0.80	0.74	0.75	0.78
	SD v2.0	0.58	0.45	0.53	0.52	0.56
	SD XL	0.42	0.38	0.44	0.40	0.40
	DALL-E2	0.46	0.35	0.35	0.31	0.28
	DALL-E3	0.46	0.38	0.41	0.26	0.39
	Midjourney v5	0.45	0.28	0.36	0.28	0.20
Real Images Accuracy		0.68	0.83	0.83	**0.89**	0.87
Generated Images Accuracy		**0.54**	0.46	0.47	0.43	0.44
Balanced Classification Accuracy		0.61	0.64	0.65	**0.66**	0.66

Table 5.13: HiFiC+SHL(g) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	Images	RF	SVM (RBF)	SVM (L)	LR	MLP
Real	RAISE-1K	0.69	0.82	0.85	0.85	0.82
Generated	Adobe Firefly	0.38	0.34	0.32	0.32	0.46
	SD v1.3	0.73	0.70	0.64	0.64	0.77
	SD v1.4	0.74	0.70	0.61	0.61	0.75
	SD v1.5	0.83	0.80	0.76	0.74	0.85
	SD v2.0	0.59	0.46	0.55	0.51	0.71
	SD XL	0.43	0.40	0.36	0.41	0.54
	DALL-E2	0.46	0.37	0.39	0.33	0.38
	DALL-E3	0.49	0.44	0.45	0.34	0.53
	Midjourney v5	0.47	0.27	0.28	0.29	0.25
Real Images Accuracy		0.69	0.82	**0.85**	**0.85**	0.82
Generated Images Accuracy		**0.56**	0.47	0.46	0.45	0.54
Balanced Classification Accuracy		0.62	0.65	0.66	0.65	**0.68**

Table 5.14: HiFiC+SHL(l) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

5.4 Spherical Harmonics Coefficients with Vision Transformer

As discussed in the section 5.2, our simple binary classifiers are unable to use inconsistencies in lighting to classify real and generated images. We use a ViT for binary classification by treating SHC for each patch as a token. Binary classification with ViT improves the performance of our proposed method both on the test set and cross-generator evaluation dataset.

5.4.1 Evaluation on Test Set

Table 5.15 presents results on the test set for SHL(l) using ViT as a binary classifier. ViT with SHL(l) gets 79% accuracy, which is higher than what we get using other binary classifiers. However, the best performance on the test set is obtained by directly using image patches. But ViT with image patches exhibit lower performance when evaluated on cross-generator dataset.

Type	**Training Method**	**ViT (Imgs)**	**ViT (Imgs+SHL(l))**	**ViT (SHL(l))**
Test Set	ViT(Scratch)	**0.83**	0.81	0.79
	ViT(Pre-trained)	**0.94**	0.92	0.79

Table 5.15: Balanced classification accuracy for ViT with SHL(l) on the test set with MLP head as last classification layer. Best performing results are marked in bold.

5.4.2 Evaluation on Cross-generator Dataset

The ViT results on the cross-generator dataset are presented in Table 5.16 and 5.17. WhenViT is trained from scratch, it gets 69% accuracy on the cross-generator dataset. The higher accuracy indicates that ViT architecture is better suited for classifying real and generated images based on lighting inconsistencies in different parts of the image.

Type	**Images**	**ViT (Imgs)**	**ViT (Imgs+ SHL(l))**	**ViT (SHL(l))**
Real	RAISE-1K	0.81	0.84	0.77
Generated	Adobe Firefly	0.35	0.41	0.51
	SD v1.3	0.68	0.71	0.70
	SD v1.4	0.66	0.68	0.66
	SD v1.5	0.81	0.80	0.84
	SD v2.0	0.55	0.59	0.74
	SD XL	0.40	0.50	0.59
	DALL-E2	0.36	0.37	0.42
	DALL-E3	0.35	0.46	0.67
	Midjourney v5	0.19	0.27	0.33
Real Images Accuracy		0.81	**0.84**	0.77
Generated Images Accuracy		0.48	0.53	**0.61**
Balanced Classification Accuracy		0.64	0.68	**0.69**

Table 5.16: ViT (trained from scratch) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

Type	**Images**	**ViT (Imgs)**	**ViT (Imgs+ SHL(l))**	**ViT (SHL(l))**
Real	RAISE-1K	0.84	0.91	0.68
Generated	Adobe Firefly	0.37	0.38	0.54
	SD v1.3	0.83	0.79	0.69
	SD v1.4	0.81	0.78	0.68
	SD v1.5	0.91	0.88	0.83
	SD v2.0	0.53	0.38	0.54
	SD XL	0.55	0.43	0.61
	DALL-E2	0.32	0.31	0.47
	DALL-E3	0.58	0.46	0.69
	Midjourney v5	0.30	0.24	0.43
Real Images Accuracy		0.84	**0.91**	0.68
Generated Images Accuracy		0.58	0.52	**0.61**
Balanced Classification Accuracy		**0.71**	0.71	0.64

Table 5.17: ViT (pre-trained) performance on cross-generator evaluation dataset. Each column shows the percentage of correctly classified images generated from the corresponding generator. Best performing results are marked in bold.

5.5 Comparison

A performance comparison of our best results with the best results from baseline methods is presented in Table 5.18. While CLIP features with simple classifiers achieve better overall balanced classification accuracy, their strong performance is mainly due to high SD v1.3 and SD v1.4 accuracy. They perform significantly worse on Firefly, DALL-E2, and Midjourney. Our ViT SHL(l) performs better on Firefly, DALL-E2, and Midjourney, although overall performance is 69%. Compared with CLIP, our ViT SHL(l) model is trained on just 10,000 images, whereas CLIP is trained on billions of images.
Finally, it is important to note that contrary to the claim that CLIP achieves high performance only with linear classifiers [21, 7], our 3-layer MLP, which includes non-linear activation and dropout layers, also achieves comparable performance.

Type	**Images**	**ViT (SHL(l))** (ours)	**CLIP + SVM (L) [7]**	**CLIP + LR [21]**	**CLIP + MLP** (ours)
Real	RAISE-1K	0.77	0.96	0.97	0.96
Generated	Adobe Firefly	0.51	0.13	0.13	0.17
	SD v1.3	0.70	0.86	0.95	0.96
	SD v1.4	0.66	0.87	0.96	0.97
	SD v1.5	0.84	0.94	0.98	0.98
	SD v2.0	0.74	0.41	0.30	0.35
	SD XL	0.59	0.42	0.33	0.42
	DALL-E2	0.42	0.15	0.15	0.18
	DALL-E3	0.67	0.83	0.66	0.79
	Midjourney v5	0.33	0.22	0.17	0.23
Real Images Accuracy		0.77	0.96	**0.97**	0.96
Generated Images Accuracy		**0.61**	0.53	0.52	0.56
Balanced Classification Accuracy		0.69	0.75	0.74	**0.76**

Table 5.18: Comparison of our proposed method with the baseline best results. Best performing results are marked in bold.

Chapter 6

Conclusion and Future Directions

Recent advancements in DL have led to the development of photo-realistic image generators. The images generated from these generators often exhibit lighting inconsistencies among various parts of the image. In this work, we address these inconsistencies by modeling the scene illumination using spherical harmonics and training various classifiers to detect them in a data-driven manner. Our best results are obtained with ViT, where we divide an image into patches, compute spherical harmonics coefficients for each patch and pass them to the ViT. Our method when trained on images generated from Stable Diffusion v1.5 only, gets 69% balanced classification accuracy when evaluated on cross-generator dataset containing 9000 images from 9 different generators.

Regarding future directions, we aim to evaluate our method by training on a larger dataset, as our current training set is limited to just 10,000 images. We also plan to test our method on a more diverse dataset from various image generators, as our current dataset includes SD v1.3, SD v1.4, and SD v1.5, which consists of images with similar quality. We used IR pipeline suggested by Yu *et al.* [18] which does not produce smooth surface normals. Recent methods *e.g.* DSINE [113] that can estimate smooth surface normals, should be used in IR pipeline.

Bibliography

[1] *Midjourney*. URL: https://www.midjourney.com (visited on 06/07/2024).

[2] OpenAI. *DALL-E3*. URL: https://openai.com/index/dall-e-3/ (visited on 06/07/2024).

[3] Adobe. *Adobe Firefly - Free generative AI for creatives*. URL: https://www.adobe.com/products/firefly.html (visited on 06/07/2024).

[4] Jonas Ricker, Dennis Assenmacher, Thorsten Holz, Asja Fischer, and Erwin Quiring. "AI-generated faces in the real world: A large-scale case study of twitter profile images". In: *ArXiv*, 2024.

[5] Jasper. *AI copilot for enterprise marketing teams*. URL: https://www.jasper.ai/ (visited on 06/11/2024).

[6] *Faceswap*. URL: https://faceswap.dev/ (visited on 06/11/2024).

[7] Davide Cozzolino, Giovanni Poggi, Riccardo Corvi, Matthias Nießner, and Luisa Verdoliva. "Raising the bar of AI-generated image detection with CLIP". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[8] Jonathan Ho, Ajay Jain, and P. Abbeel. "Denoising diffusion probabilistic models". In: *Neural Information Processing Systems*, 2020.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Neural Information Processing Systems*, 2014.

[10] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. The MIT Press, 2023.

[11] Hany Farid and Mary J. Bravo. "Image forensic analyses that elude the human visual system". In: *Electronic Imaging*, 2010.

[12] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. "Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, pp. 7327–7347, 2021.

[13] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. "Conditional image generation with PixelCNN decoders". In: *Neural Information Processing Systems*, 2016.

[14] Diederik P. Kingma and Max Welling. “Auto-encoding variational bayes”. In: *International Conference on Learning Representations*, 2013.

[15] Danilo Jimenez Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International Conference on Machine Learning*, 2015.

[16] Prafulla Dhariwal and Alex Nichol. “Diffusion models beat GANs on image synthesis”. In: *Neural Information Processing Systems*, 2021.

[17] Quentin Bammey. “Synthbuster: Towards detection of diffusion model generated images”. In: *IEEE Open Journal of Signal Processing* 5, pp. 1–9, 2024.

[18] Ye Yu and W. Smith. “Outdoor inverse rendering from a single image using multiview self-supervision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, pp. 3659–3675, 2021.

[19] Zheng Zeng, Valentin Deschaintre, Iliyan Georgiev, Yannick Hold-Geoffroy, Yiwei Hu, Fujun Luan, Ling-Qi Yan, and Milos Hasan. “RGB $\leftrightarrow$ X: Image decomposition and synthesis using material-and lighting-aware diffusion models”. In: *ArXiv*, 2024.

[20] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. “Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and SVBRDF from a single image”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2472–2481, 2019.

[21] Utkarsh Ojha, Yuheng Li, and Yong Jae Lee. “Towards universal fake image detectors that generalize across generative models”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24480–24489, 2023.

[22] David C. Epstein, Ishan Jain, Oliver Wang, and Richard Zhang. “Online detection of AI-generated images”. In: *IEEE/CVF International Conference on Computer Vision Workshops*, pp. 382–392, 2023.

[23] Sergey Sinitsa and Ohad Fried. “Deep image fingerprint: Accurate and low budget synthetic image detector”. In: *IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024.

[24] Keshigeyan Chandrasegaran, Ngoc-Trung Tran, Alexander Binder, and Ngai-Man Cheung. “Discovering transferable forensic features for CNN-generated images detection”. In: *European Conference on Computer Vision* 13675, pp. 671–689, 2022.

[25] Ayush Sarkar, Hanlin Mai, Amitabh Mahapatra, Svetlana Lazebnik, D. A. Forsyth, and Anand Bhattad. “Shadows don’t lie and lines can’t bend! Generative models don’t know projective geometry...for now”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[26] Diangarti Bhalang Tariang, Riccardo Corvi, Davide Cozzolino, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. “Synthetic image verification in the vra of generative artificial intelligence: what works and what isn’t there yet”. In: *IEEE Security & Privacy* 22, pp. 37–49, 2024.

[27] Falko Matern, Christian Riess, and Marc Stamminger. "Exploiting visual artifacts to expose deepfakes and face manipulations". In: *IEEE Winter Applications of Computer Vision Workshops*, pp. 83–92, 2019.

[28] Matyás Bohácek and Hany Farid. "A geometric and photometric exploration of GAN and diffusion synthesized faces". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 874–883, 2023.

[29] Eric Kee, James F. O'Brien, and Hany Farid. "Exposing photo manipulation with inconsistent shadows". In: *ACM Transactions on Graphics* 32, 28:1–28:12, 2013.

[30] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A. Efros. "CNN-generated images are surprisingly easy to spot... for now". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8692–8701, 2019.

[31] Yonghyun Jeong, Doyeon Kim, Youngmin Ro, Pyounggeon Kim, and Jongwon Choi. "FingerprintNet: Synthesized fingerprints for generated image Detection". In: *European Conference on Computer Vision* 13674, pp. 76–94, 2022.

[32] Ravi Ramamoorthi and Pat Hanrahan. "An efficient representation for irradiance environment maps". In: *International Conference on Computer Graphics and Interactive Techniques*, 2001.

[33] *"Over the top". A composite photograph, originally known as "A hop over"*. URL: `https://www.awm.gov.au/collection/C194879` (visited on 06/01/2024).

[34] Diederik P. Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *CoRR*, 2014.

[35] Aston Zhang, Zachary Chase Lipton, Mu Li, and Alexander J. Smola. "Dive into deep learning". In: *Journal of the American College of Radiology*, 2020.

[36] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *ArXiv*, 2012.

[37] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning* 37, pp. 448–456, 2015.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86, pp. 2278–2324, 1998.

[40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, K. Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[41] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *CoRR*, 2014.

[42] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2015.

[43] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in Neural Information Processing Systems*, 2017.

[44] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *International Conference on Learning Representations*, 2021.

[45] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari S. Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Grégoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. "A cookbook of self-supervised learning". In: *ArXiv*, 2023.

[46] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*, 2021.

[47] David Ha, Andrew M. Dai, and Quoc V. Le. "Hypernetworks". In: *International Conference on Learning Representations*, 2016.

[48] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training GANs". In: *Neural Information Processing Systems*, 2016.

[49] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. "GANs trained by a two time-scale update rule converge to a local Nash equilibrium". In: *Neural Information Processing Systems*, 2017.

[50] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *International Conference on Learning Representations*, 2016.

[51] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *ArXiv*, 2014.

[52] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. "Progressive growing of GANs for improved quality, stability, and variation". In: *International Conference on Learning Representations*, 2018.

[53] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-resolution image synthesis with latent diffusion models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10674–10685, 2021.

[54] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. "Your classifier is secretly an energy based model and you should treat it like one". In: *International Conference on Learning Representations*, 2020.

[55] Bo Liu, Fan Yang, Xiuli Bi, Bin Xiao, Weisheng Li, and Xinbo Gao. "Detecting generated images by real images". In: *European Conference on Computer Vision* 13674, pp. 95–110, 2022.

[56] Yichi Zhang and Xiaogang Xu. "Diffusion noise feature: Accurate and fast generated image detection". In: *CoRR*, 2023.

[57] Yan Ju, Shan Jia, Lipeng Ke, Hongfei Xue, Koki Nagano, and Siwei Lyu. "Fusing global and local features for generalized AI-synthesized image detection". In: *International Conference on Image Processing*, pp. 3465–3469, 2022.

[58] Davide Cozzolino, Justus Thies, Andreas Rössler, Christian Riess, Matthias Nießner, and Luisa Verdoliva. "ForensicTransfer: Weakly-supervised domain adaptation for forgery detection". In: *CoRR*, 2018.

[59] Eric Kee, James F. O'Brien, and Hany Farid. "Exposing photo manipulation from shading and shadows". In: *ACM Transactions on Graphics* 33, pp. 1–21, 2014.

[60] Micah K. Johnson and Hany Farid. "Exposing digital forgeries in complex lighting environments". In: *IEEE Transactions on Information Forensics and Security* 2, pp. 450–461, 2007.

[61] Christian Riess and Elli Angelopoulou. "Scene illumination as an indicator of image manipulation". In: *Information Hiding*, 2010.

[62] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. "Detecting and simulating artifacts in GAN fake images". In: *IEEE International Workshop on Information Forensics and Security*, pp. 1–6, 2019.

[63] Joel Frank, Thorsten Eisenhofer, Lea Schönherr, Asja Fischer, Dorothea Kolossa, and Thorsten Holz. "Leveraging frequency analysis for deep fake image recognition". In: *International Conference on Machine Learning*, pp. 3247–3258, 2020.

[64] Zhendong Wang, Jianmin Bao, Wengang Zhou, Weilun Wang, Hezhen Hu, Hong Chen, and Houqiang Li. "DIRE for diffusion-generated image detection". In: *IEEE/CVF International Conference on Computer Vision*, pp. 22388–22398, 2023.

[65] Riccardo Corvi, Davide Cozzolino, Giada Zingarini, Giovanni Poggi, Koki Nagano, and Luisa Verdoliva. "On the detection of synthetic images generated by diffusion models". In: *International Conference on Acoustics, Speech and Signal Processing*, pp. 1–5, 2023.

[66] Pantelis Dogoulis, Giorgos Kordopatis-Zilos, Ioannis Kompatsiaris, and Symeon Papadopoulos. "Improving synthetically generated image detection in cross-concept settings". In: *International Workshop on Multimedia AI against Disinformation*, pp. 28–35, 2023.

[67] Micah K. Johnson and Hany Farid. "Exposing digital forgeries by detecting inconsistencies in lighting". In: *Workshop on Multimedia & Security*, 2005.

[68] Paul E. Debevec. "Rendering with natural light". In: *International Conference on Computer Graphics and Interactive Techniques*, 1998.

[69] Robert Green. "Spherical harmonic lighting: The gritty details". In: *Archives of the Game Developers Conference*, 2003.

[70] Harry G. Barrow and Jay M. Tenenbaum. *Recovering intrinsic scene characteristics from images*. SRI International, 1978.

[71] Edwin Herbert Land and John J. McCann. "Lightness and retinex theory". In: *Journal of the Optical Society of America* 61 1, pp. 1–11, 1971.

[72] Jonathan T. Barron and Jitendra Malik. "Shape, illumination, and reflectance from shading". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, pp. 1670–1687, 2015.

[73] Sean Bell, Kavita Bala, and Noah Snavely. "Intrinsic images in the wild". In: *ACM Transactions on Graphics* 33, pp. 1–12, 2014.

[74] Roger Baker Grosse, Micah K. Johnson, Edward H. Adelson, and William T. Freeman. "Ground truth dataset and baseline evaluations for intrinsic image algorithms". In: *IEEE International Conference on Computer Vision*, pp. 2335–2342, 2009.

[75] Balazs Kovacs, Sean Bell, Noah Snavely, and Kavita Bala. "Shading annotations in the wild". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 850–859, 2017.

[76] Isabella Liu, Ling Chen, Ziyang Fu, Liwen Wu, Haian Jin, Zhong Li, Chin Ming Ryan Wong, Yi Xu, Ravi Ramamoorthi, Zexiang Xu, and Hao Su. "OpenIllumination: A multi-illumination dataset for inverse rendering evaluation on real objects". In: *ArXiv*, 2023.

[77] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas A. Funkhouser. "Semantic scene completion from a single depth image". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 190–198, 2016.

[78] Soumyadip Sengupta, Jinwei Gu, Kihwan Kim, Guilin Liu, David W. Jacobs, and Jan Kautz. "Neural inverse rendering of an indoor scene from a single image". In: *IEEE/CVF International Conference on Computer Vision*, pp. 8597–8606, 2019.

[79] Peter Kocsis, Vincent Sitzmann, and Matthias Niessner. "Intrinsic image diffusion for indoor single-view material estimation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[80] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. "LAION-5B: An open large-scale dataset for training next generation image-text models". In: *Neural Information Processing Systems*, 2022.

[81] Romain Beaumont. *Clip Retrieval: Easily compute clip embeddings and build a clip retrieval system with them.* 2022. URL: `https://github.com/rom1504/clip-retrieval` (visited on 06/19/2024).

[82] *A Python perceptual image hashing module.* URL: `https://github.com/JohannesBuchner/imagehash` (visited on 06/06/2024).

[83] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. "GIT: A generative image-to-text transformer for vision and language". In: *Transactions on Machine Learning Research* 2022, 2022.

[84] *GIT (GenerativeImage2Text), large-sized, fine-tuned on COCO.* URL: `https://huggingface.co/microsoft/git-large-r-coco` (visited on 06/06/2024).

[85] Runway ML. *Stable Diffusion v1.5.* URL: `https://huggingface.co/runwayml/stable-diffusion-v1-5` (visited on 06/06/2024).

[86] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. "Pseudo numerical methods for diffusion models on manifolds". In: 2022.

[87] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. *Diffusers: State-of-the-art diffusion models.* 2022. URL: `https://github.com/huggingface/diffusers` (visited on 06/06/2024).

[88] Jonathan Ho. "Classifier-free diffusion guidance". In: *ArXiv*, 2022.

[89] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. "GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models". In: *International Conference on Machine Learning*, 2021.

[90] *OpenAI.* URL: `https://openai.com/` (visited on 06/07/2024).

[91] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical text-conditional image generation with CLIP latents". In: *ArXiv*, 2022. URL: `https://api.semanticscholar.org/CorpusID:248097655`.

[92] James Betker, Gabriel Goh, Li Jing, TimBrooks, Jianfeng Wang, Linjie Li, LongOuyang, JuntangZhuang, JoyceLee, YufeiGuo, WesamManassra, PrafullaDhariwal, CaseyChu, YunxinJiao, and Aditya Ramesh. "Improving Image Generation with Better Captions". In: 2023.

[93] Midjourney. *Midjourney V5.* URL: `https://docs.midjourney.com/docs/model-version-5` (visited on 06/20/2024).

[94] CompVis LMU Munich. *Stable Diffusion v1.* URL: https://github.com/CompVis/stable-diffusion (visited on 06/20/2024).

[95] Stability AI. *Stable Diffusion v2.* URL: https://huggingface.co/stabilityai/stable-diffusion-2 (visited on 06/20/2024).

[96] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. *OpenCLIP*. 2021. URL: https://github.com/mlfoundations/open_clip (visited on 06/20/2024).

[97] Stability AI. *Stable Diffusion XL.* URL: https://huggingface.co/docs/diffusers/en/using-diffusers/sdxl (visited on 06/20/2024).

[98] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. "eDiff-I: Text-to-image diffusion models with an ensemble of expert denoisers". In: *ArXiv*, 2022. URL: https://api.semanticscholar.org/CorpusID:253254800.

[99] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. "RAISE: A raw images dataset for digital image forensics". In: *ACM Multimedia Systems Conference*, 2015.

[100] Yu Yee. *Implementation of "Outdoor inverse rendering from a single image using multiview self-supervision".* URL: https://github.com/YeeU/InverseRenderNet_v2 (visited on 06/08/2024).

[101] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *2016 European Conference on Computer Vision*, 2016.

[102] Zhengqi Li and Noah Snavely. "MegaDepth: Learning single-view depth prediction from internet photos". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2041–2050, 2018.

[103] Ravi Ramamoorthi and Pat Hanrahan. "A signal-processing framework for inverse rendering". In: *International Conference on Computer Graphics and Interactive Techniques*, 2001.

[104] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine learning in Python". In: *Machine Learning Research* 12, pp. 2825–2830, 2011.

[105] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *International Conference on Learning Representations*, 2017.

[106] PyTorch. *ReduceLROnPlateau 2014; PyTorch 2.3 documentation.* URL: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html (visited on 06/10/2024).

[107] Google. *Vision transformer (base-sized model).* URL: https://huggingface.co/google/vit-base-patch32-224-in21k (visited on 06/10/2024).

[108] Aude Oliva and Antonio Torralba. "Modeling the shape of the scene: A holistic representation of the spatial envelope". In: *International Journal of Computer Vision* 42.3, pp. 145–175, May 2001.

[109] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. "High-fidelity generative image compression". In: *Advances in Neural Information Processing Systems*, 2020.

[110] OpenAI. *CLIP: Contrastive language-image pretraining.* URL: https://github.com/openai/CLIP (visited on 06/16/2024).

[111] *Spatial envelope.* URL: https://people.csail.mit.edu/torralba/code/spatialenvelope/ (visited on 06/13/2024).

[112] *Pytorch Implementation of HiFiC.* URL: https://github.com/Justin-Tan/high-fidelity-generative-compression (visited on 06/21/2024).

[113] Gwangbin Bae and Andrew J Davison. "Rethinking inductive biases for surface normal estimation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9535–9545, 2024.