IMPORTING LIBRARIES

In [1]:
```
import os
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
import torch.optim as optim
from torchvision.utils import save_image
from tqdm import tqdm
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from pdf2image import convert_from_path
import docx
from transformers import BertTokenizer, BertModel
import torch.nn.functional as F
from torchmetrics.image.kid import KernelInceptionDistance
import cv2
from skimage.metrics import structural_similarity as ssim
import lpips
from scipy.fft import fft2, fftshift
```

CONVERTING PDF PAGES TO IMAGES

In [2]:
```
PDF_DIR = "/home/aniketj/GSOC_TASK3/PDFs/"  # Directory containing PDFs
IMAGE_DIR = "/home/aniketj/GSOC_TASK3/IMAGES/"  # Output directory for image

os.makedirs(IMAGE_DIR, exist_ok=True)
```

In [3]:
```
def pdf_to_images(pdf_path, output_folder, dpi=250):
    images = convert_from_path(pdf_path, dpi=dpi, fmt="jpeg")
    image_paths = []

    for i, img in enumerate(images):
        img = img.convert("RGB")
        img_path = os.path.join(output_folder, f"{os.path.basename(pdf_path)
        img.save(img_path, "JPEG", quality=85)
        image_paths.append(img_path)

    return image_paths
```

In [4]:
```
# Process all PDFs in the folder
for pdf in os.listdir(PDF_DIR):
    if pdf.endswith(".pdf"):
        pdf_to_images(os.path.join(PDF_DIR, pdf), IMAGE_DIR, dpi=250)

print(" PDF to Image Conversion Done!")
```

```
/home/aniketj/anaconda3/envs/soc/lib/python3.10/site-packages/PIL/Image.py:3
402: DecompressionBombWarning: Image size (147000000 pixels) exceeds limit o
f 89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
```

```
PDF to Image Conversion Done!
```

TEXT EMBEDDINGS

In [5]:
```python
# Load BERT model and tokenizer for text embeddings
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
bert_model = BertModel.from_pretrained("bert-base-uncased").to(device)
```

In [6]:
```python
# Function to extract text from docx files
def extract_text_from_docx(docx_path):
    doc = docx.Document(docx_path)
    text = " ".join([p.text for p in doc.paragraphs])
    return text
```

In [ ]:
```python
# Function to generate text embeddings
def get_text_embedding(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=T
    with torch.no_grad():
        outputs = bert_model(**inputs)
    return outputs.last_hidden_state.mean(dim=1)
```

In [8]:
```python
# Dataset class incorporating both images and text embeddings
class TextImageDataset(Dataset):
    def __init__(self, img_dir, txt_dir, transform=None):
        self.img_paths = [os.path.join(img_dir, f) for f in os.listdir(img_d
        self.txt_paths = {os.path.splitext(f)[0]: os.path.join(txt_dir, f) f
        self.transform = transform

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img_path = self.img_paths[idx]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)

        doc_name = os.path.splitext(os.path.basename(img_path))[0]
        text_embedding = torch.zeros((1, 768))
        if doc_name in self.txt_paths:
            text = extract_text_from_docx(self.txt_paths[doc_name])
            text_embedding = get_text_embedding(text)

        return image, text_embedding.squeeze(0)
```

In [9]:
```python
# Image transformations
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

CREATING DATALOADER

In [10]:
```
# Load dataset
dataset = TextImageDataset(img_dir="/home/aniketj/GSOC_TASK3/IMAGES", txt_di
dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

print("Dataset loaded with images and text embeddings")
```

```
Dataset loaded with images and text embeddings
```

GENERATOR MODEL

In [11]:
```
# Generator model
class Generator(nn.Module):
    def __init__(self, input_channels=3, text_dim=768, output_channels=3, nu
        super(Generator, self).__init__()
        self.text_fc = nn.Linear(text_dim, 256 * 256)  # Embed text into ima
        self.initial = nn.Sequential(
            nn.Conv2d(input_channels + 1, 64, kernel_size=7, stride=1, paddi
            nn.ReLU(inplace=True)
        )
        self.residual_blocks = nn.Sequential(*[ResidualBlock(64) for _ in ra
        self.final = nn.Sequential(
            nn.Conv2d(64, output_channels, kernel_size=7, stride=1, padding=
            nn.Tanh()
        )

    def forward(self, x, text):
        text_map = self.text_fc(text).view(-1, 1, 256, 256)  # Reshape to ma
        x = torch.cat([x, text_map], dim=1)  # Concatenate text representati
        x = self.initial(x)
        x = self.residual_blocks(x)
        return self.final(x)
```

RESIDUAL BLOCKS

In [12]:
```
# Residual block for generator
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1
            nn.ReLU(inplace=True),
            nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1
        )

    def forward(self, x):
        return x + self.block(x)
```

DISCRIMINATOR MODEL

In [13]:
```
# Discriminator model
class Discriminator(nn.Module):
    def __init__(self, input_channels=3):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
```

```
            nn.Conv2d(input_channels, 64, kernel_size=4, stride=2, padding=1
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 1, kernel_size=4, stride=1, padding=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)
```

In [28]:
```
# Function to generate text-based images
def generate_text_image(model, test_image_path):
    model.eval()
    image = Image.open(test_image_path).convert("RGB")
    image = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        fake_image = model(image, torch.zeros(1, 768).to(device))

    save_image(fake_image, "generated_renaissance_image_2.png")
    print("Saved: generated_renaissance_image_2.png")
```

In [15]:
```
# Initialize models
G = Generator().to(device)
D = Discriminator().to(device)
```

In [16]:
```
# Loss and optimizers
criterion_GAN = nn.BCELoss()
criterion_cycle = nn.L1Loss()
optimizer_G = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
optimizer_D = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))
```

In [17]:
```
GEN_IMAGE_DIR = "/home/aniketj/GSOC_TASK3/GENERATED_IMAGES/"  # Output direc

os.makedirs(GEN_IMAGE_DIR, exist_ok=True)
```

GAN TRAINING

In [18]:
```
# Training loop
num_epochs = 150
for epoch in range(num_epochs):
    for i, (real_text_images, text_embeddings) in enumerate(dataloader):
        real_text_images, text_embeddings = real_text_images.to(device), tex

        # Generate fake images
        fake_text_images = G(real_text_images, text_embeddings)

        # Train Discriminator
        optimizer_D.zero_grad()
        real_loss = criterion_GAN(D(real_text_images), torch.ones_like(D(rea
        fake_loss = criterion_GAN(D(fake_text_images.detach()), torch.zeros_
```

```
        D_loss = (real_loss + fake_loss) / 2
        D_loss.backward()
        optimizer_D.step()

        # Train Generator
        optimizer_G.zero_grad()
        G_loss = criterion_GAN(D(fake_text_images), torch.ones_like(D(fake_t
                 criterion_cycle(fake_text_images, real_text_images) * 10
        G_loss.backward()
        optimizer_G.step()

        if i % 10 == 0:
            print(f"Epoch [{epoch+1}/{num_epochs}], D Loss: {D_loss.item()},

    save_image(fake_text_images, f"/home/aniketj/GSOC_TASK3/GENERATED_IMAGES
```

```
/home/aniketj/anaconda3/envs/soc/lib/python3.10/site-packages/PIL/Image.py:3
402: DecompressionBombWarning: Image size (147000000 pixels) exceeds limit o
f 89478485 pixels, could be decompression bomb DOS attack.
  warnings.warn(
```

```
Epoch [1/150], D Loss: 0.6993781328201294, G Loss: 8.0424222946167
Epoch [2/150], D Loss: 0.690129280090332, G Loss: 2.3335325717926025
Epoch [3/150], D Loss: 0.6770015954971313, G Loss: 1.814490795135498
Epoch [4/150], D Loss: 0.6641082167625427, G Loss: 1.9355055093765259
Epoch [5/150], D Loss: 0.6796083450317383, G Loss: 1.7592716217041016
Epoch [6/150], D Loss: 0.7169135212898254, G Loss: 2.310405731201172
Epoch [7/150], D Loss: 0.6789689064025879, G Loss: 1.5329442024230957
Epoch [8/150], D Loss: 0.6925828456878662, G Loss: 1.613318920135498
Epoch [9/150], D Loss: 0.6681864261627197, G Loss: 1.9327009916305542
Epoch [10/150], D Loss: 0.6927521824836731, G Loss: 1.5753633975982666
Epoch [11/150], D Loss: 0.6924472451210022, G Loss: 1.3305492401123047
Epoch [12/150], D Loss: 0.6917015314102173, G Loss: 1.3783529996871948
Epoch [13/150], D Loss: 0.6877474784851074, G Loss: 1.2524375915527344
Epoch [14/150], D Loss: 0.6911436319351196, G Loss: 1.3102660179138184
Epoch [15/150], D Loss: 0.6931610107421875, G Loss: 1.5839762687683105
Epoch [16/150], D Loss: 0.688393771648407, G Loss: 1.712237000465393
Epoch [17/150], D Loss: 0.6936396956443787, G Loss: 1.3520724773406982
Epoch [18/150], D Loss: 0.6892629861831665, G Loss: 1.2422826290130615
Epoch [19/150], D Loss: 0.6896337270736694, G Loss: 1.392808198928833
Epoch [20/150], D Loss: 0.6996279358863831, G Loss: 1.4504268169403076
Epoch [21/150], D Loss: 0.691075325012207, G Loss: 1.2847319841384888
Epoch [22/150], D Loss: 0.6960354447364807, G Loss: 1.9684563875198364
Epoch [23/150], D Loss: 0.693596601486206, G Loss: 1.5445804595947266
Epoch [24/150], D Loss: 0.6904991865158081, G Loss: 1.3125910758972168
Epoch [25/150], D Loss: 0.6928995251655579, G Loss: 1.2896130084991455
Epoch [26/150], D Loss: 0.6928995251655579, G Loss: 1.1797854900360107
Epoch [27/150], D Loss: 0.6946316957473755, G Loss: 1.1806817054748535
Epoch [28/150], D Loss: 0.6960066556930542, G Loss: 1.5968356132507324
Epoch [29/150], D Loss: 0.6900591254234314, G Loss: 1.5121464729309082
Epoch [30/150], D Loss: 0.691204309463501, G Loss: 1.3948395252227783
Epoch [31/150], D Loss: 0.6952701807022095, G Loss: 1.3178925514221191
Epoch [32/150], D Loss: 0.6913191080093384, G Loss: 1.4478306770324707
Epoch [33/150], D Loss: 0.6931216716766357, G Loss: 1.0881359577178955
Epoch [34/150], D Loss: 0.6928051710128784, G Loss: 1.127772331237793
Epoch [35/150], D Loss: 0.6915382146835327, G Loss: 1.1163924932479858
Epoch [36/150], D Loss: 0.692013144493103, G Loss: 1.4665125608444214
Epoch [37/150], D Loss: 0.6915181875228882, G Loss: 1.084275245666504
Epoch [38/150], D Loss: 0.6920992136001587, G Loss: 1.3996565341949463
Epoch [39/150], D Loss: 0.692341148853302, G Loss: 1.0908784866333008
Epoch [40/150], D Loss: 0.6921097636222839, G Loss: 1.1129167079925537
Epoch [41/150], D Loss: 0.692991316318512, G Loss: 1.0448170900344849
Epoch [42/150], D Loss: 0.6933317184448242, G Loss: 1.148383617401123
Epoch [43/150], D Loss: 0.6932662725448608, G Loss: 1.276801347732544
Epoch [44/150], D Loss: 0.6929571628570557, G Loss: 1.0081322193145752
Epoch [45/150], D Loss: 0.6923885345458984, G Loss: 1.2005995512008667
Epoch [46/150], D Loss: 0.6927738785743713, G Loss: 1.1416685581207275
Epoch [47/150], D Loss: 0.6922097206115723, G Loss: 1.062135934829712
Epoch [48/150], D Loss: 0.6917047500610352, G Loss: 1.4239720106124878
Epoch [49/150], D Loss: 0.6888521909713745, G Loss: 1.4752683639526367
Epoch [50/150], D Loss: 0.6915081739425659, G Loss: 1.1882991790771484
Epoch [51/150], D Loss: 0.6907509565353394, G Loss: 1.2243728637695312
Epoch [52/150], D Loss: 0.6908639669418335, G Loss: 1.0724279880523682
Epoch [53/150], D Loss: 0.6930254697799683, G Loss: 1.2647022008895874
Epoch [54/150], D Loss: 0.6918848752975464, G Loss: 1.0284734964370728
Epoch [55/150], D Loss: 0.6902098059654236, G Loss: 0.9847891330718994
Epoch [56/150], D Loss: 0.692560613155365, G Loss: 1.1670918464660645
```

```
Epoch [57/150], D Loss: 0.6916823387145996, G Loss: 1.2018651962280273
Epoch [58/150], D Loss: 0.6893607974052429, G Loss: 1.1908857822418213
Epoch [59/150], D Loss: 0.6914240717887878, G Loss: 1.3030281066894531
Epoch [60/150], D Loss: 0.6951260566711426, G Loss: 1.1950560808181763
Epoch [61/150], D Loss: 0.6935329437255859, G Loss: 1.21556556224823
Epoch [62/150], D Loss: 0.6942203640937805, G Loss: 1.1050599813461304
Epoch [63/150], D Loss: 0.6932662725448608, G Loss: 1.2564442157745361
Epoch [64/150], D Loss: 0.6898341178894043, G Loss: 0.9803879261016846
Epoch [65/150], D Loss: 0.6936739683151245, G Loss: 1.01882803440094
Epoch [66/150], D Loss: 0.6914469003677368, G Loss: 1.0285570621490479
Epoch [67/150], D Loss: 0.6915956735610962, G Loss: 1.2119724750518799
Epoch [68/150], D Loss: 0.6954803466796875, G Loss: 1.1231098175048828
Epoch [69/150], D Loss: 0.6911877393722534, G Loss: 1.066701054573059
Epoch [70/150], D Loss: 0.6937174797058105, G Loss: 0.966423511505127
Epoch [71/150], D Loss: 0.6918582916259766, G Loss: 0.9633148312568665
Epoch [72/150], D Loss: 0.6911354064941406, G Loss: 1.2633094787597656
Epoch [73/150], D Loss: 0.7013876438140869, G Loss: 1.2824761867523193
Epoch [74/150], D Loss: 0.6921581029891968, G Loss: 1.0775575637817383
Epoch [75/150], D Loss: 0.6921303272247314, G Loss: 1.0010707378387451
Epoch [76/150], D Loss: 0.6974023580551147, G Loss: 1.284813642501831
Epoch [77/150], D Loss: 0.693490743637085, G Loss: 1.0756895542144775
Epoch [78/150], D Loss: 0.6932411789894104, G Loss: 1.1490278244018555
Epoch [79/150], D Loss: 0.6923561692237854, G Loss: 1.3440114259719849
Epoch [80/150], D Loss: 0.6936154365539551, G Loss: 1.149569034576416
Epoch [81/150], D Loss: 0.6923296451568604, G Loss: 1.0143895149230957
Epoch [82/150], D Loss: 0.6915127038955688, G Loss: 0.9950001835823059
Epoch [83/150], D Loss: 0.6903378963470459, G Loss: 0.9984863996505737
Epoch [84/150], D Loss: 0.6918012499809265, G Loss: 0.97883141040802
Epoch [85/150], D Loss: 0.693076491355896, G Loss: 1.274521827697754
Epoch [86/150], D Loss: 0.6927927732467651, G Loss: 1.1054677963256836
Epoch [87/150], D Loss: 0.6914563179016113, G Loss: 0.9580438137054443
Epoch [88/150], D Loss: 0.6952284574508667, G Loss: 0.9401296973228455
Epoch [89/150], D Loss: 0.6920078992843628, G Loss: 1.0278964042663574
Epoch [90/150], D Loss: 0.6972206830978394, G Loss: 1.374962329864502
Epoch [91/150], D Loss: 0.694277286529541, G Loss: 1.1412066221237183
Epoch [92/150], D Loss: 0.6939228773117065, G Loss: 1.1272056102752686
Epoch [93/150], D Loss: 0.6920822262763977, G Loss: 0.923160970211029
Epoch [94/150], D Loss: 0.6911333203315735, G Loss: 1.0706830024719238
Epoch [95/150], D Loss: 0.6992026567459106, G Loss: 1.049478530883789
Epoch [96/150], D Loss: 0.6930973529815674, G Loss: 0.9528816342353821
Epoch [97/150], D Loss: 0.6922893524169922, G Loss: 0.9589226245880127
Epoch [98/150], D Loss: 0.6949846744537354, G Loss: 1.2414181232452393
Epoch [99/150], D Loss: 0.6917235255241394, G Loss: 0.9247637391090393
Epoch [100/150], D Loss: 0.6935724020004272, G Loss: 1.0743494033813477
Epoch [101/150], D Loss: 0.6928441524505615, G Loss: 0.9754431247711182
Epoch [102/150], D Loss: 0.6915782690048218, G Loss: 1.0882909297943115
Epoch [103/150], D Loss: 0.6917580366134644, G Loss: 1.0171265602111816
Epoch [104/150], D Loss: 0.6968774199485779, G Loss: 1.1933743953704834
Epoch [105/150], D Loss: 0.692076563835144, G Loss: 0.9586048722267151
Epoch [106/150], D Loss: 0.692243218421936, G Loss: 0.9778721928596497
Epoch [107/150], D Loss: 0.6916087865829468, G Loss: 0.953521728515625
Epoch [108/150], D Loss: 0.6932977437973022, G Loss: 1.072891116142273
Epoch [109/150], D Loss: 0.6898584365844727, G Loss: 1.128706932067871
Epoch [110/150], D Loss: 0.6977704763412476, G Loss: 1.1749377250671387
Epoch [111/150], D Loss: 0.6924178600311279, G Loss: 0.9834259152412415
Epoch [112/150], D Loss: 0.6960018873214722, G Loss: 1.2776813507080078
```

```
Epoch [113/150], D Loss: 0.6962084174156189, G Loss: 1.0876293182373047
Epoch [114/150], D Loss: 0.6923364400863647, G Loss: 0.9985641241073608
Epoch [115/150], D Loss: 0.6913880109786987, G Loss: 1.0399441719055176
Epoch [116/150], D Loss: 0.6949803829193115, G Loss: 1.0001592636108398
Epoch [117/150], D Loss: 0.6925556659698486, G Loss: 0.8958827257156372
Epoch [118/150], D Loss: 0.6907081604003906, G Loss: 0.9890016317367554
Epoch [119/150], D Loss: 0.693528413772583, G Loss: 1.047576665878296
Epoch [120/150], D Loss: 0.6917115449905396, G Loss: 0.9634324312210083
Epoch [121/150], D Loss: 0.6896682977676392, G Loss: 1.1624863147735596
Epoch [122/150], D Loss: 0.6943548917770386, G Loss: 1.0638015270233154
Epoch [123/150], D Loss: 0.6962268352508545, G Loss: 1.1723092794418335
Epoch [124/150], D Loss: 0.6897368431091309, G Loss: 1.1690328121185303
Epoch [125/150], D Loss: 0.6916564702987671, G Loss: 1.0269850492477417
Epoch [126/150], D Loss: 0.6961235404014587, G Loss: 1.0897157192230225
Epoch [127/150], D Loss: 0.6931151151657104, G Loss: 0.9299312233924866
Epoch [128/150], D Loss: 0.6925361156463623, G Loss: 0.9338114261627197
Epoch [129/150], D Loss: 0.6906739473342896, G Loss: 0.923717737197876
Epoch [130/150], D Loss: 0.6909713745117188, G Loss: 1.0002751350402832
Epoch [131/150], D Loss: 0.6926806569099426, G Loss: 0.9666247963905334
Epoch [132/150], D Loss: 0.693144679069519, G Loss: 1.0488810539245605
Epoch [133/150], D Loss: 0.6927071809768677, G Loss: 1.0137498378753662
Epoch [134/150], D Loss: 0.6931244134902954, G Loss: 0.9355742931365967
Epoch [135/150], D Loss: 0.6924868822097778, G Loss: 0.9582609534263611
Epoch [136/150], D Loss: 0.6921550035476685, G Loss: 0.9209317564964294
Epoch [137/150], D Loss: 0.6925610899925232, G Loss: 0.9052624702453613
Epoch [138/150], D Loss: 0.6916686296463013, G Loss: 1.2075587511062622
Epoch [139/150], D Loss: 0.6930692791938782, G Loss: 1.031320571899414
Epoch [140/150], D Loss: 0.6924314498901367, G Loss: 0.9830909371376038
Epoch [141/150], D Loss: 0.6929696202278137, G Loss: 0.913377046585083
Epoch [142/150], D Loss: 0.6929445266723633, G Loss: 0.8821373581886292
Epoch [143/150], D Loss: 0.6928671598434448, G Loss: 0.9559253454208374
Epoch [144/150], D Loss: 0.6926151514053345, G Loss: 0.9779489040374756
Epoch [145/150], D Loss: 0.6923778057098389, G Loss: 1.0865740776062012
Epoch [146/150], D Loss: 0.6919565200805664, G Loss: 1.1802290678024292
Epoch [147/150], D Loss: 0.6916155815124512, G Loss: 0.9222742319107056
Epoch [148/150], D Loss: 0.6953754425048828, G Loss: 1.1403318643569946
Epoch [149/150], D Loss: 0.6926787495613098, G Loss: 1.0653043985366821
Epoch [150/150], D Loss: 0.6933372616767883, G Loss: 1.0529581308364868
```

SAVING THE GENERATOR AND DISCRIMINATOR MODEL

In [19]:
```
# Save trained models
torch.save(G.state_dict(), "/home/aniketj/GSOC_TASK3/generator_renaissance.p
torch.save(D.state_dict(), "/home/aniketj/GSOC_TASK3/discriminator_renaissar
print("Models saved successfully!")
```

```
Models saved successfully!
```

In [20]:
```
# Generate text-based image
generate_text_image(G, "/home/aniketj/GSOC_TASK3/TEST_IMAGES/99.jpg")
```

```
Saved: generated_renaissance_image.png
```

EVALUATION METRICS : Structural Similarity Index Measure (SSIM)

In [21]:
```python
def calculate_ssim(img1_path, img2_path):
    img1 = cv2.imread(img1_path, cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread(img2_path, cv2.IMREAD_GRAYSCALE)

    if img1 is None:
        raise FileNotFoundError(f"Could not load image: {img1_path}")
    if img2 is None:
        raise FileNotFoundError(f"Could not load image: {img2_path}")

    # Resize images if they have different shapes
    if img1.shape != img2.shape:
        img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

    score, _ = ssim(img1, img2, full=True)
    return score
```

EVALUATION METRICS : Peak Signal-to-Noise Ratio (PSNR)

In [ ]:
```python
# PSNR
def calculate_psnr(img1_path, img2_path):
    img1 = cv2.imread(img1_path)
    img2 = cv2.imread(img2_path)

    if img1.shape != img2.shape:
        img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))

    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return float('inf')
    return 20 * np.log10(255.0 / np.sqrt(mse))
```

In [23]:
```python
# Function to load and preprocess an image for LPIPS
def load_image(image_path):
    img = cv2.imread(image_path)  # Load image (BGR format)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB
    img = cv2.resize(img, (256, 256))  # Resize to 256x256
    img = np.transpose(img, (2, 0, 1)) / 255.0 * 2 - 1  # Normalize to [-1,1
    img = torch.tensor(img, dtype=torch.float32).unsqueeze(0)  # Convert to
    return img
```

In [24]:
```python
# Function to generate images
def generate_images(input_dir, output_dir):
    image_files = [f for f in os.listdir(input_dir) if f.endswith(('.jpg', '

    for img_name in tqdm(image_files, desc="Generating Images"):
        img_path = os.path.join(input_dir, img_name)
        image = Image.open(img_path).convert("RGB")
        image = transform(image).unsqueeze(0).to(device)

        with torch.no_grad():
            generated_image = G(image, torch.zeros(1, 768).to(device))

        save_path = os.path.join(output_dir, f"{img_name}")
        save_image(generated_image, save_path)
```

```
    print(f"Generated images saved in: {output_dir}")
```

In [25]:
```
G.eval()
```

Out[25]:
```
Generator(
  (text_fc): Linear(in_features=768, out_features=65536, bias=True)
  (initial): Sequential(
    (0): Conv2d(4, 64, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): ReLU(inplace=True)
  )
  (residual_blocks): Sequential(
    (0): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (1): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (2): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (3): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (4): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (5): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
```

```
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
  )
  (final): Sequential(
    (0): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): Tanh()
  )
)
```

In [24]:
```
# Load the trained generator model
G = Generator().to(device)
G.load_state_dict(torch.load("/home/aniketj/GSOC_TASK3/generator_renaissance
G.eval()  # Set model to evaluation mode
```

```
/tmp/ipykernel_1146755/787309532.py:3: FutureWarning: You are using `torch.l
oad` with `weights_only=False` (the current default value), which uses the d
efault pickle module implicitly. It is possible to construct malicious pickl
e data which will execute arbitrary code during unpickling (See https://gith
ub.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more detai
ls). In a future release, the default value for `weights_only` will be flipp
ed to `True`. This limits the functions that could be executed during unpick
ling. Arbitrary objects will no longer be allowed to be loaded via this mode
unless they are explicitly allowlisted by the user via `torch.serialization.
add_safe_globals`. We recommend you start setting `weights_only=True` for an
y use case where you don't have full control of the loaded file. Please open
an issue on GitHub for any issues related to this experimental feature.
  G.load_state_dict(torch.load("/home/aniketj/GSOC_TASK3/generator_renaissan
ce.pth", map_location=device))
```

Out[24]:
```
Generator(
  (text_fc): Linear(in_features=768, out_features=65536, bias=True)
  (initial): Sequential(
    (0): Conv2d(4, 64, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): ReLU(inplace=True)
  )
  (residual_blocks): Sequential(
    (0): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (1): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (2): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (3): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (4): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
    (5): ResidualBlock(
      (block): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
```

```
        (1): ReLU(inplace=True)
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      )
    )
  )
  (final): Sequential(
    (0): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): Tanh()
  )
)
```

In [27]:
```
# Generate text-based image
generate_text_image(G, "/home/aniketj/GSOC_TASK3/TEST_IMAGES/779.jpg")
```

```
Saved: generated_renaissance_image_1.png
```

In [29]:
```
# Generate text-based image
generate_text_image(G, "/home/aniketj/GSOC_TASK3/TEST_IMAGES/318.jpg")
```

```
Saved: generated_renaissance_image_2.png
```

In [30]:
```
ssim_score = calculate_ssim("/home/aniketj/GSOC_TASK3/CODE/generated_renaiss
print("SSIM Score:", ssim_score)
```

```
SSIM Score: 0.9175970416388124
```

In [31]:
```
psnr_score = calculate_psnr("/home/aniketj/GSOC_TASK3/TEST_IMAGES/99.jpg", "
print("PSNR Score:" ,psnr_score)
```

```
PSNR Score: 31.516438178369164
```

EVALUATION METRICS : Learned Perceptual Image Patch Similarity (LPIPS)

In [32]:
```
# Load LPIPS model (AlexNet backbone)
loss_fn = lpips.LPIPS(net='alex')
```

```
Setting up [LPIPS] perceptual loss: trunk [alex], v[0.1], spatial [off]
/home/aniketj/anaconda3/envs/soc/lib/python3.10/site-packages/torchvision/mo
dels/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated si
nce 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/home/aniketj/anaconda3/envs/soc/lib/python3.10/site-packages/torchvision/mo
dels/_utils.py:223: UserWarning: Arguments other than a weight enum or `None
` for 'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=AlexNet_Weights.IMAGE
NET1K_V1`. You can also use `weights=AlexNet_Weights.DEFAULT` to get the mos
t up-to-date weights.
  warnings.warn(msg)
Loading model from: /home/aniketj/anaconda3/envs/soc/lib/python3.10/site-pac
kages/lpips/weights/v0.1/alex.pth
```

```
/home/aniketj/anaconda3/envs/soc/lib/python3.10/site-packages/lpips/lpips.p
y:107: FutureWarning: You are using `torch.load` with `weights_only=False`
(the current default value), which uses the default pickle module implicitl
y. It is possible to construct malicious pickle data which will execute arbi
trary code during unpickling (See https://github.com/pytorch/pytorch/blob/ma
in/SECURITY.md#untrusted-models for more details). In a future release, the
default value for `weights_only` will be flipped to `True`. This limits the
functions that could be executed during unpickling. Arbitrary objects will n
o longer be allowed to be loaded via this mode unless they are explicitly al
lowlisted by the user via `torch.serialization.add_safe_globals`. We recomme
nd you start setting `weights_only=True` for any use case where you don't ha
ve full control of the loaded file. Please open an issue on GitHub for any i
ssues related to this experimental feature.
  self.load_state_dict(torch.load(model_path, map_location='cpu'), strict=Fa
lse)
```

In [33]:
```python
# Function to load and preprocess an image for LPIPS
def load_image(image_path):
    img = cv2.imread(image_path)  # Load image (BGR format)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB
    img = cv2.resize(img, (256, 256))  # Resize to 256x256
    img = np.transpose(img, (2, 0, 1)) / 255.0 * 2 - 1  # Normalize to [-1,1
    img = torch.tensor(img, dtype=torch.float32).unsqueeze(0)  # Convert to
    return img
```

In [34]:
```python
# Load fake and real images
real_image = load_image("/home/aniketj/GSOC_TASK3/TEST_IMAGES/99.jpg")  # Re
fake_image = load_image("/home/aniketj/GSOC_TASK3/CODE/generated_renaissance

# Compute LPIPS distance (lower is better)
lpips_score = loss_fn(real_image, fake_image)
print("LPIPS Score:", lpips_score.item())
```

```
LPIPS Score: 0.10925519466400146
```

In [36]:
```python
# Define directories
test_input_dir = "/home/aniketj/GSOC_TASK3/TEST_IMAGES"  # Directory contair
test_output_dir = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES"  # Direct
os.makedirs(test_output_dir, exist_ok=True)  # Create output directory if it
```

In [38]:
```python
generate_images(test_input_dir, test_output_dir)
```

```
Generating Images: 100%|██████████| 28/28 [00:06<00:00,  4.41it/s]
Generated images saved in: /home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES
```

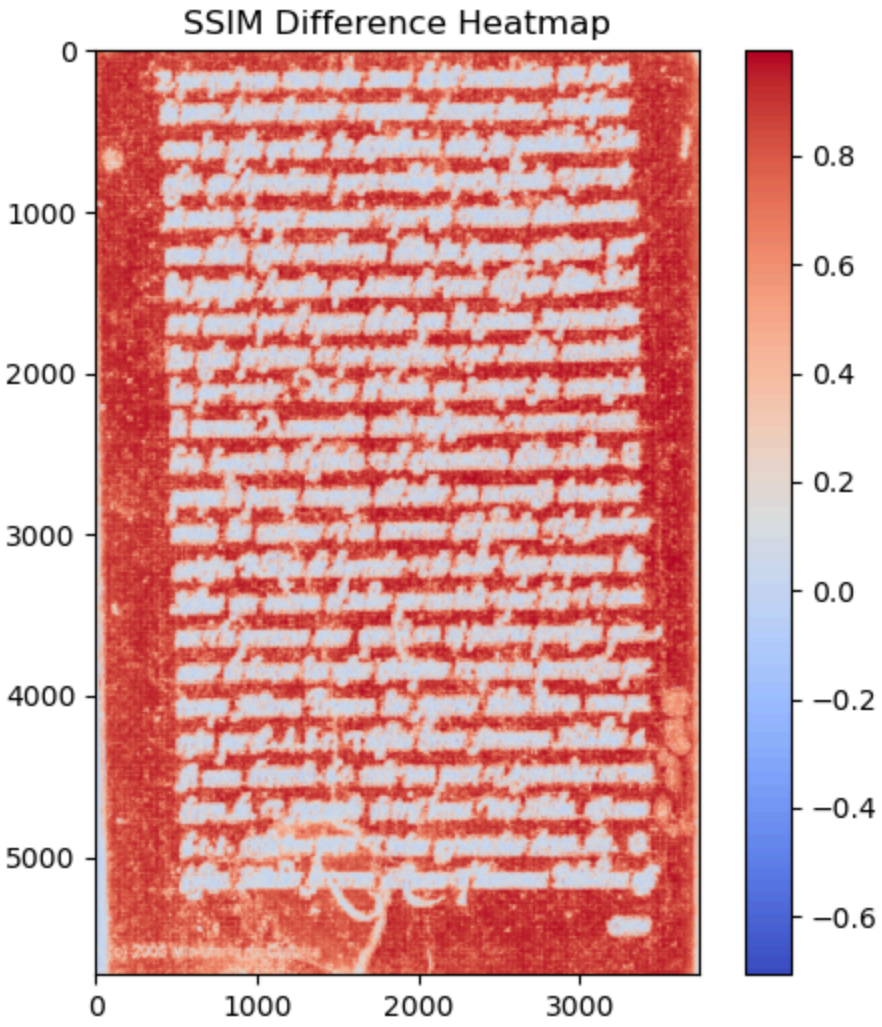
In [39]:
```python
# Load images
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/9.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/9.jpg"
```

In [40]:
```python
real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)
```

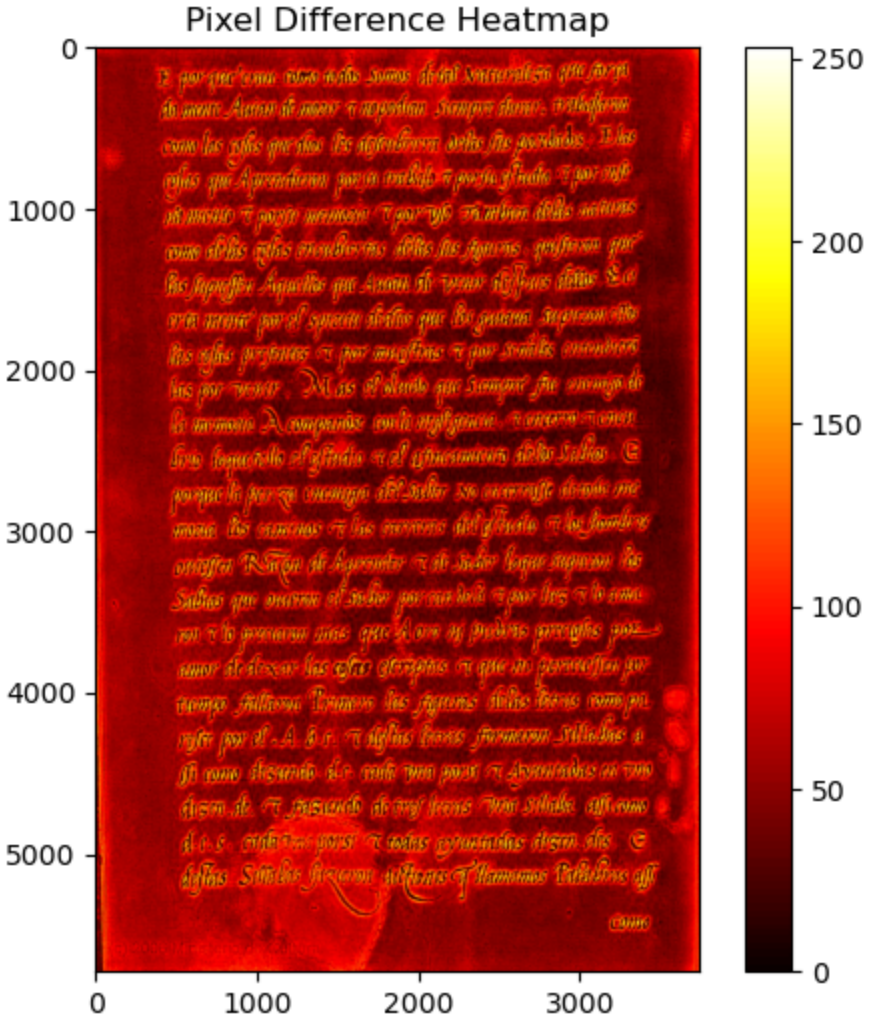
In [41]:
```python
# Resize images to match dimensions
fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))
```

VISUALIZATIONS

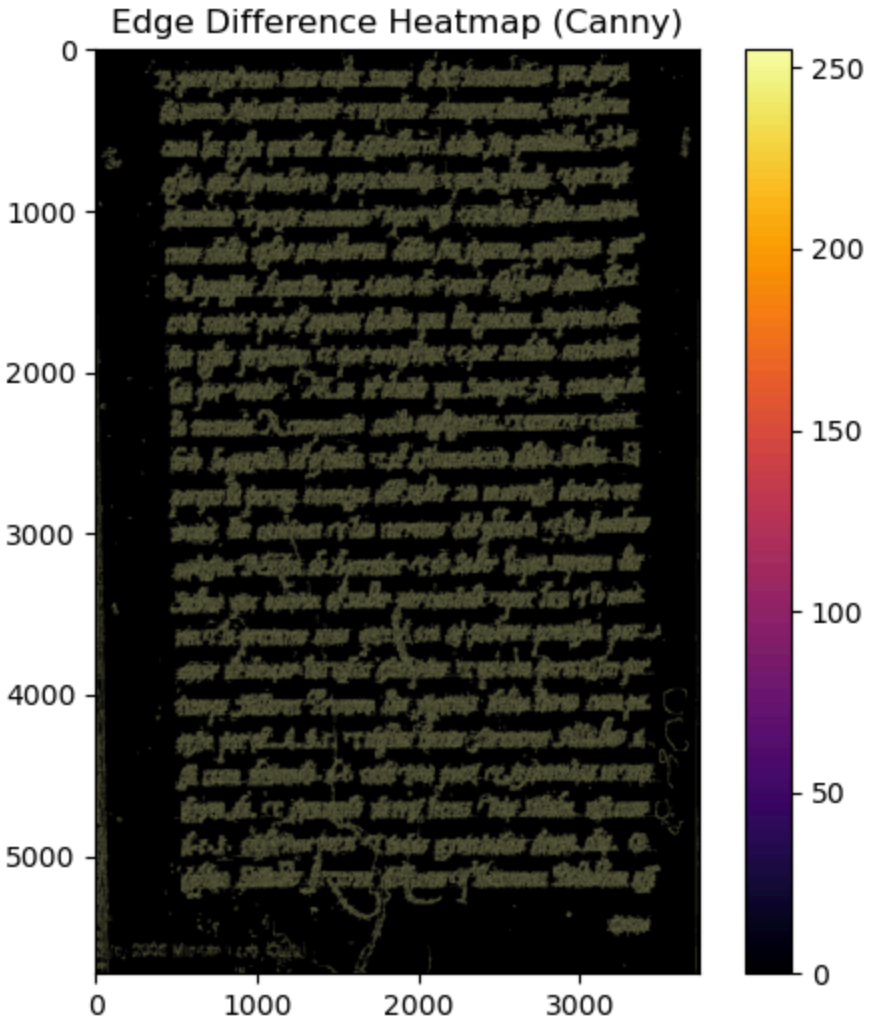
In [43]:
```
# SSIM Difference Heatmap
ssim_score, ssim_diff = ssim(real_img, fake_img, full=True)
plt.figure(figsize=(6, 6))
plt.imshow(ssim_diff, cmap='coolwarm')
plt.colorbar()
plt.title("SSIM Difference Heatmap")
plt.show()
```



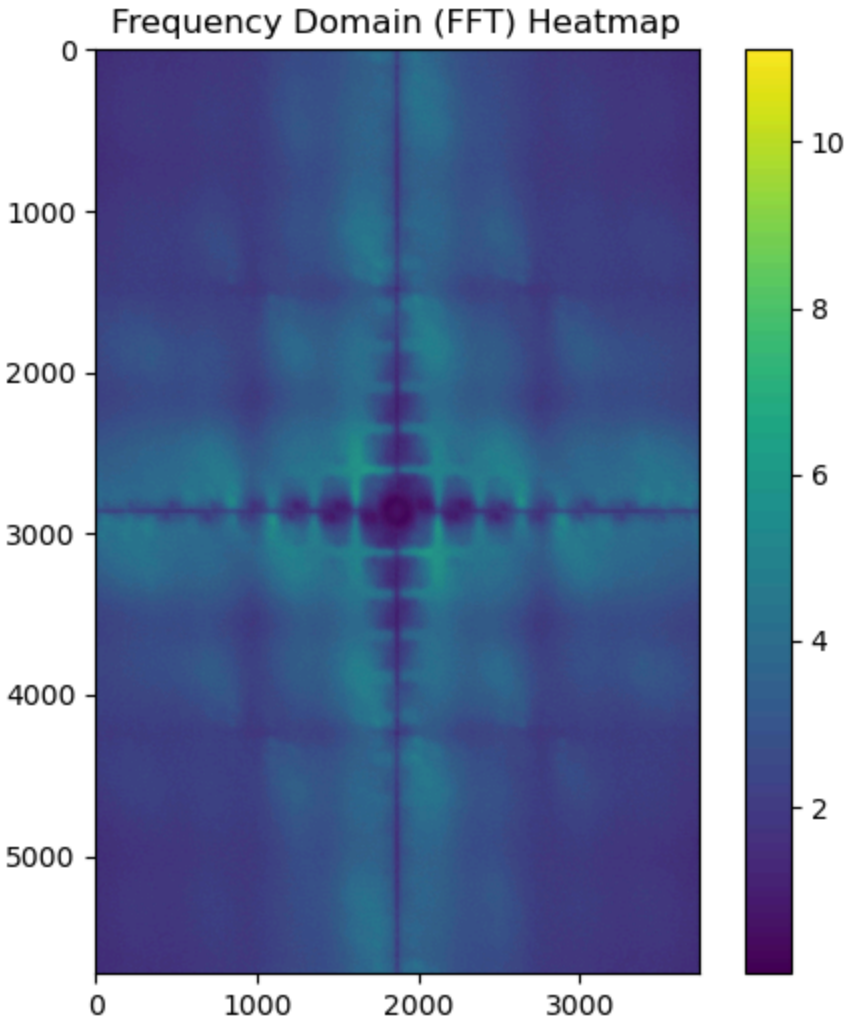
In [44]:
```
#  Pixel-Wise Difference Heatmap
pixel_diff = cv2.absdiff(real_img, fake_img)
plt.figure(figsize=(6, 6))
plt.imshow(pixel_diff, cmap='hot')
plt.colorbar()
plt.title("Pixel Difference Heatmap")
plt.show()
```



In [45]:
```
#  Edge Detection Heatmap
real_edges = cv2.Canny(real_img, 50, 150)
fake_edges = cv2.Canny(fake_img, 50, 150)
edge_diff = cv2.absdiff(real_edges, fake_edges)
plt.figure(figsize=(6, 6))
plt.imshow(edge_diff, cmap='inferno')
plt.colorbar()
plt.title("Edge Difference Heatmap (Canny)")
plt.show()
```

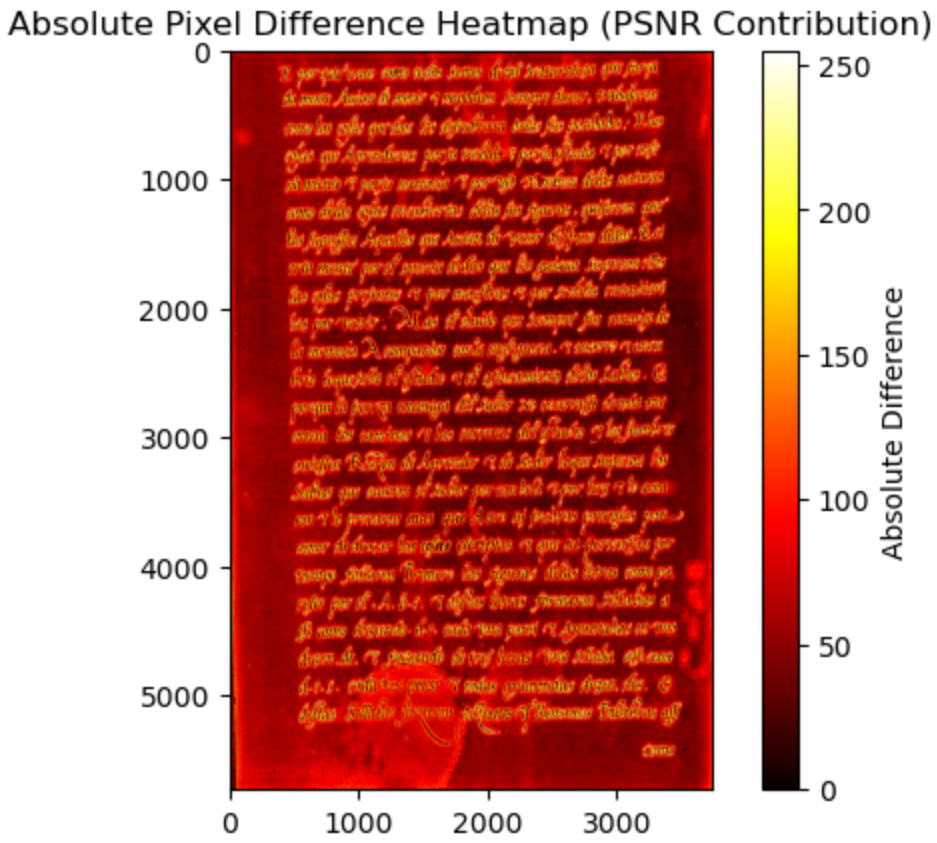


In [46]:
```
# Frequency Domain (FFT) Heatmap
real_fft = np.log(np.abs(fftshift(fft2(real_img))) + 1)
fake_fft = np.log(np.abs(fftshift(fft2(fake_img))) + 1)
fft_diff = np.abs(real_fft - fake_fft)
plt.figure(figsize=(6, 6))
plt.imshow(fft_diff, cmap='viridis')
plt.colorbar()
plt.title("Frequency Domain (FFT) Heatmap")
plt.show()
```

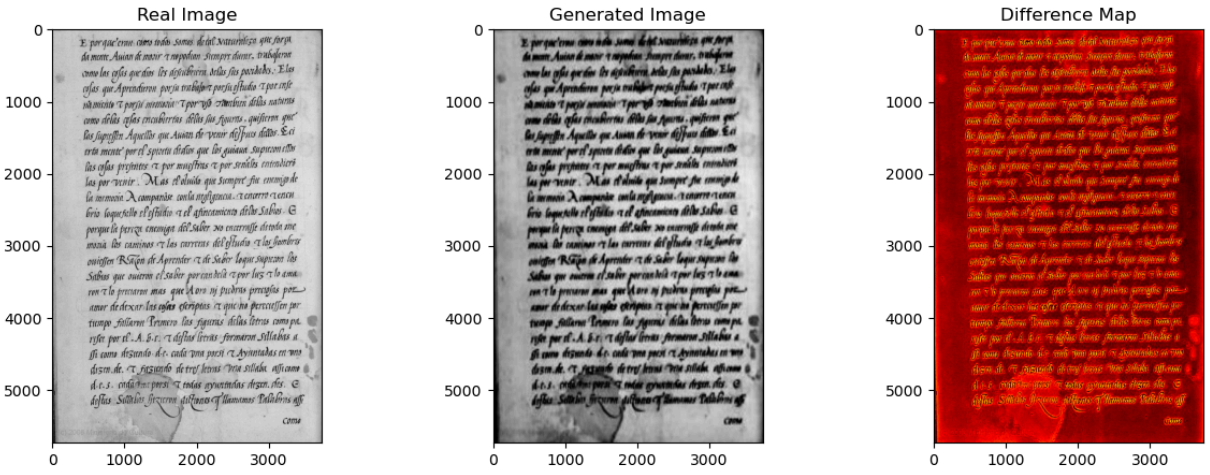


In [47]:
```python
# Compute absolute difference map
abs_diff_map = np.abs(real_img - fake_img)

# Plot heatmap
plt.imshow(abs_diff_map, cmap="hot")
plt.colorbar(label="Absolute Difference")
plt.title("Absolute Pixel Difference Heatmap (PSNR Contribution)")
plt.show()
```



In [66]:
```
# 1. Image Comparisons
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



In [84]:
```
psnr_scores = []

# Directories
real_imgs = "/home/aniketj/GSOC_TASK3/TEST_IMAGES"
generated_imgs = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES"
```

```
# Get sorted list of image files (ensure names match)
real_images = sorted(os.listdir(real_imgs))
generate_images = sorted(os.listdir(generated_imgs))

# Print corresponding image paths
for real, generated in zip(real_images, generate_images):
    real_img_path = os.path.join(real_imgs, real)
    generated_img_path = os.path.join(generated_imgs, generated)

    # Load fake and real images
    real_image = load_image(real_img_path)  # Replace with your real image p
    fake_image = load_image(generated_img_path)  # Replace with your fake (g
    psnr_score = calculate_psnr(real_img_path, generated_img_path)
    psnr_scores.append(psnr_score)
```

In [85]:
```
len(psnr_scores)
```

Out[85]: 28

In [86]:
```
psnr_scores.pop()
```

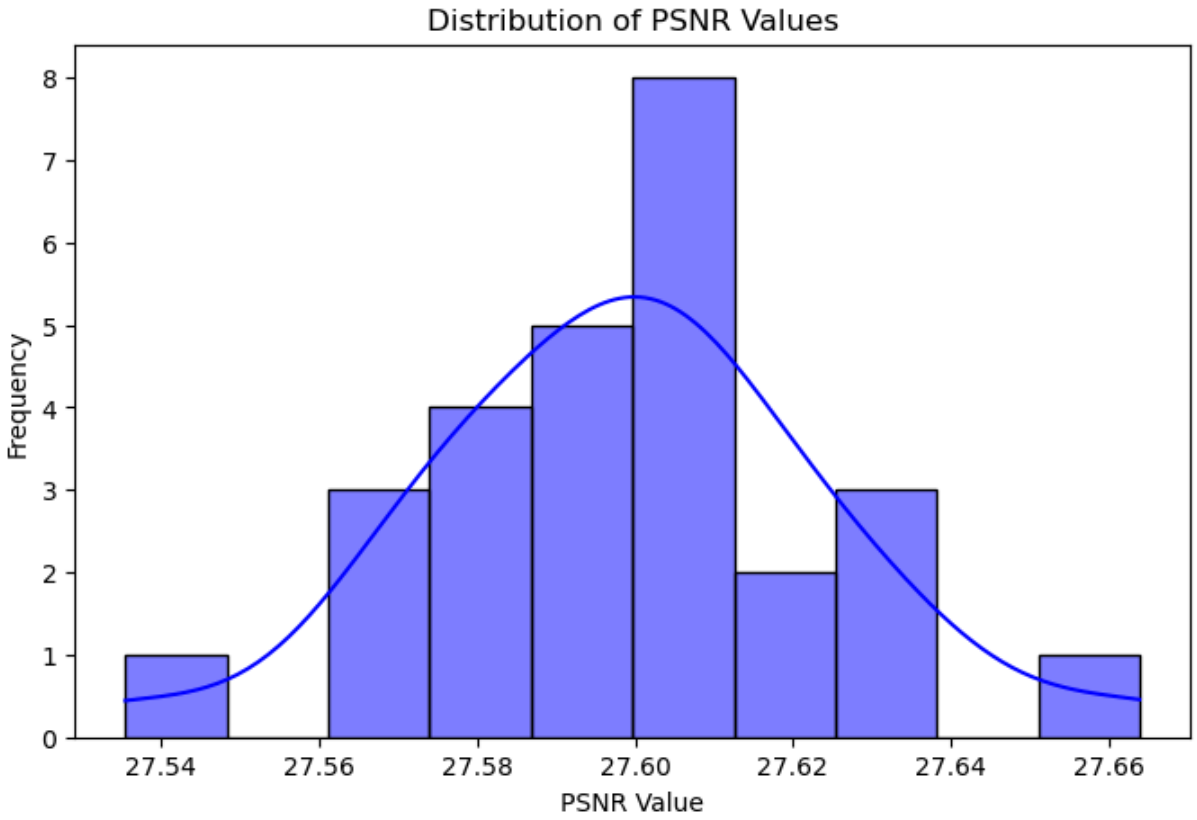
Out[86]: np.float64(30.81662819805057)

In [87]:
```
len(psnr_scores)
```

Out[87]: 27

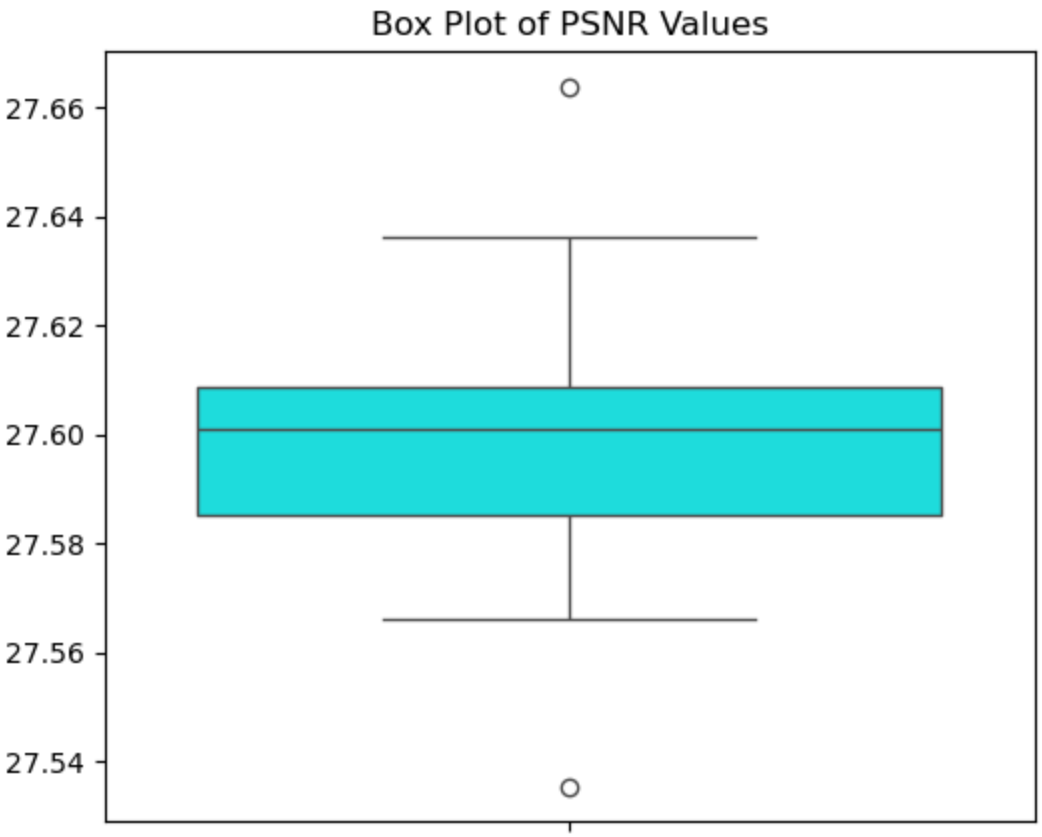
In [91]:
```
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.histplot(psnr_scores, bins=10, kde=True, color='blue')
plt.xlabel("PSNR Value")
plt.ylabel("Frequency")
plt.title("Distribution of PSNR Values")
plt.show()
psnr_distribution_output_path = "/home/aniketj/GSOC_TASK3/PLOTS/psnr_distrib
plt.savefig(psnr_distribution_output_path, bbox_inches="tight")
```



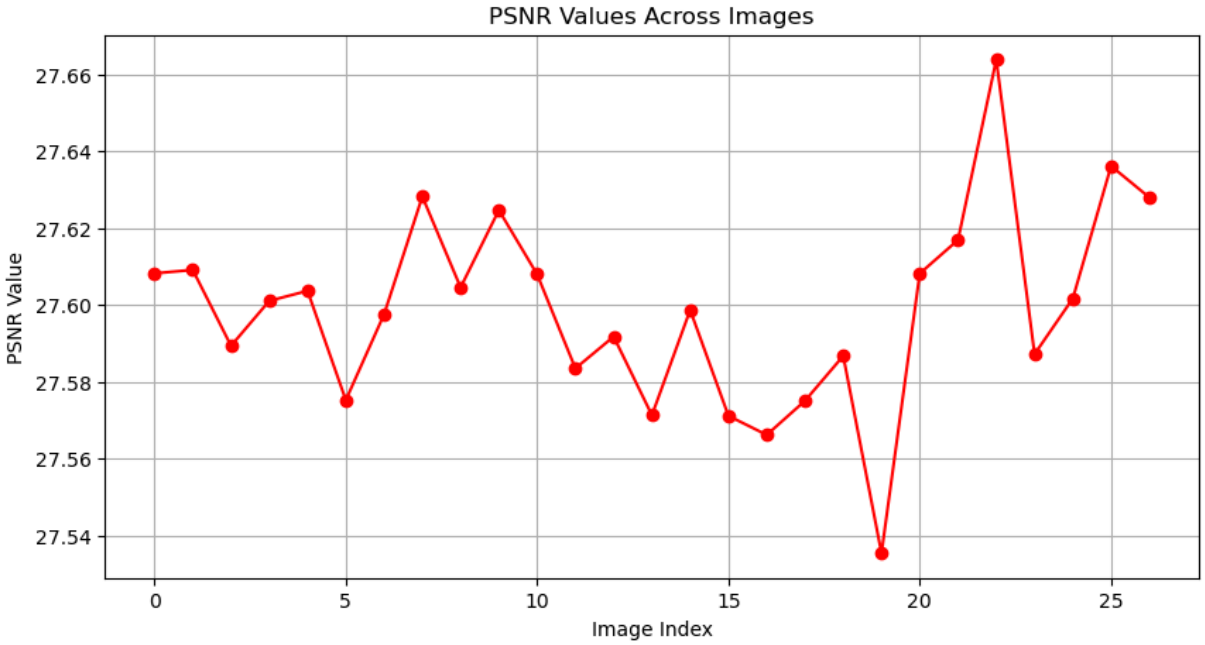

```
<Figure size 640x480 with 0 Axes>
```

In [92]:
```
plt.figure(figsize=(6, 5))
sns.boxplot(psnr_scores, color='cyan')
plt.title("Box Plot of PSNR Values")
plt.show()
psnr_outliers_output_path = "/home/aniketj/GSOC_TASK3/PLOTS/psnr_outliers.pr
plt.savefig(psnr_outliers_output_path, bbox_inches="tight")
```



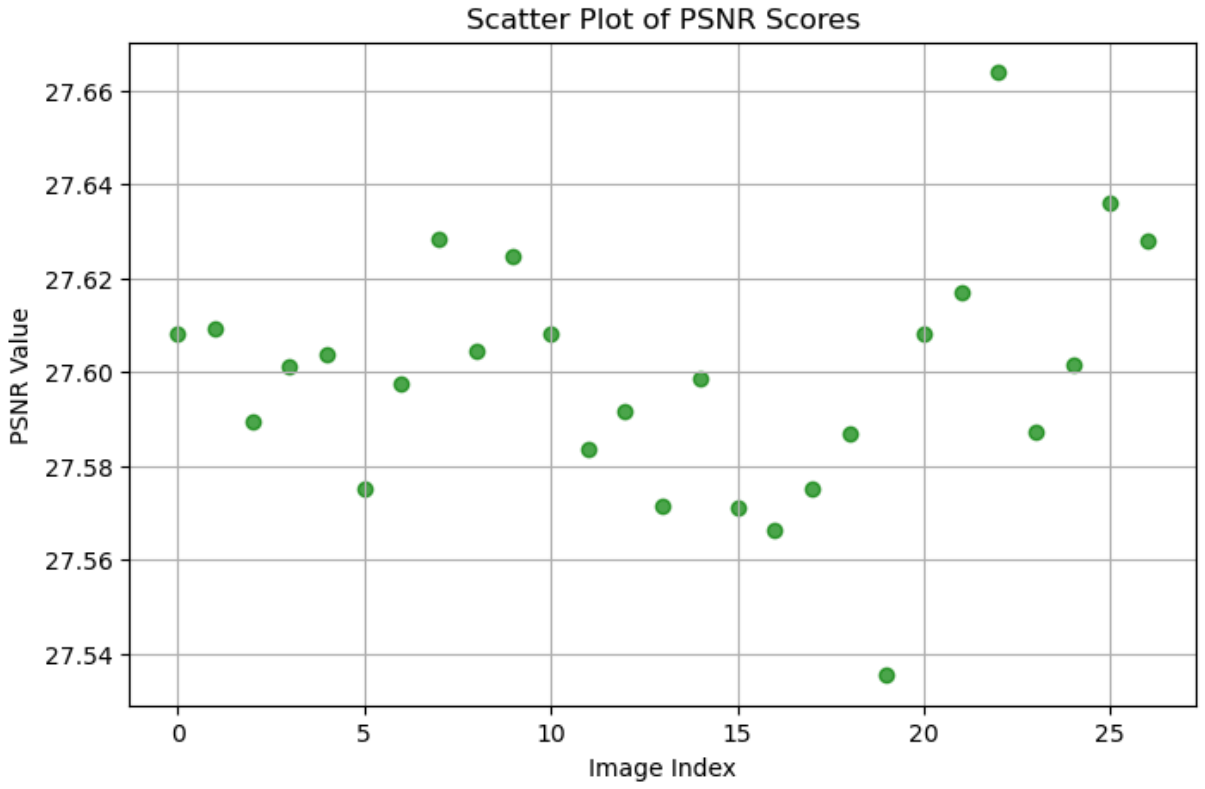
<Figure size 640x480 with 0 Axes>

In [93]:
```
plt.figure(figsize=(10, 5))
plt.plot(psnr_scores, marker='o', linestyle='-', color='red')
plt.xlabel("Image Index")
plt.ylabel("PSNR Value")
plt.title("PSNR Values Across Images")
plt.grid()
plt.show()
psnr_trend_output_path = "/home/aniketj/GSOC_TASK3/PLOTS/psnr_trend.png"
plt.savefig(psnr_trend_output_path, bbox_inches="tight")
```



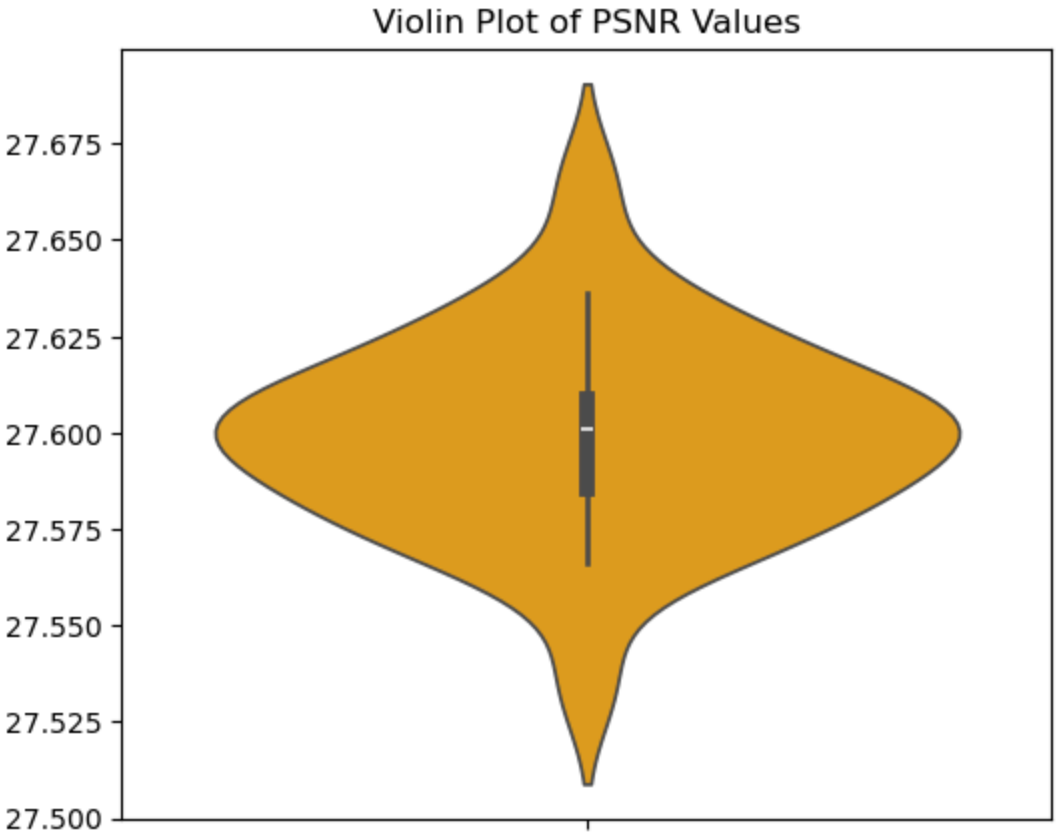
<Figure size 640x480 with 0 Axes>

In [95]:
```
plt.figure(figsize=(8, 5))
plt.scatter(range(len(psnr_scores)), psnr_scores, color='green', alpha=0.7)
plt.xlabel("Image Index")
plt.ylabel("PSNR Value")
plt.title("Scatter Plot of PSNR Scores")
plt.grid()
plt.show()
psnr_scatter_output_path = "/home/aniketj/GSOC_TASK3/PLOTS/psnr_scatter.png"
plt.savefig(psnr_scatter_output_path, bbox_inches="tight")
```



```
<Figure size 640x480 with 0 Axes>
```

In [97]:
```
plt.figure(figsize=(6, 5))
sns.violinplot(psnr_scores, color='orange')
plt.title("Violin Plot of PSNR Values")
plt.show()
psnr_violin_output_path = "/home/aniketj/GSOC_TASK3/PLOTS/psnr_violin.png"
plt.savefig(psnr_violin_output_path, bbox_inches="tight")
```



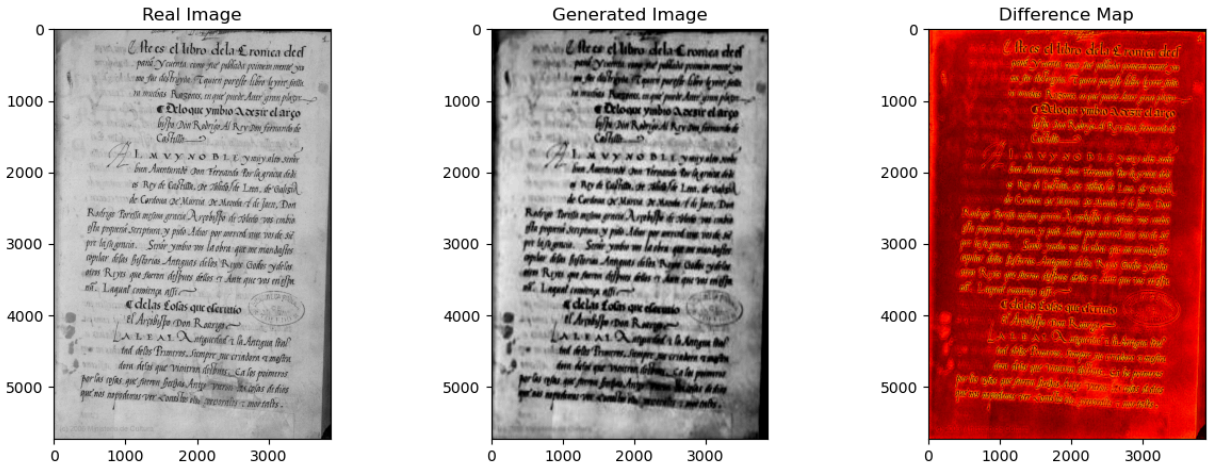

```
<Figure size 640x480 with 0 Axes>
```

In [100…
```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/8.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/8.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



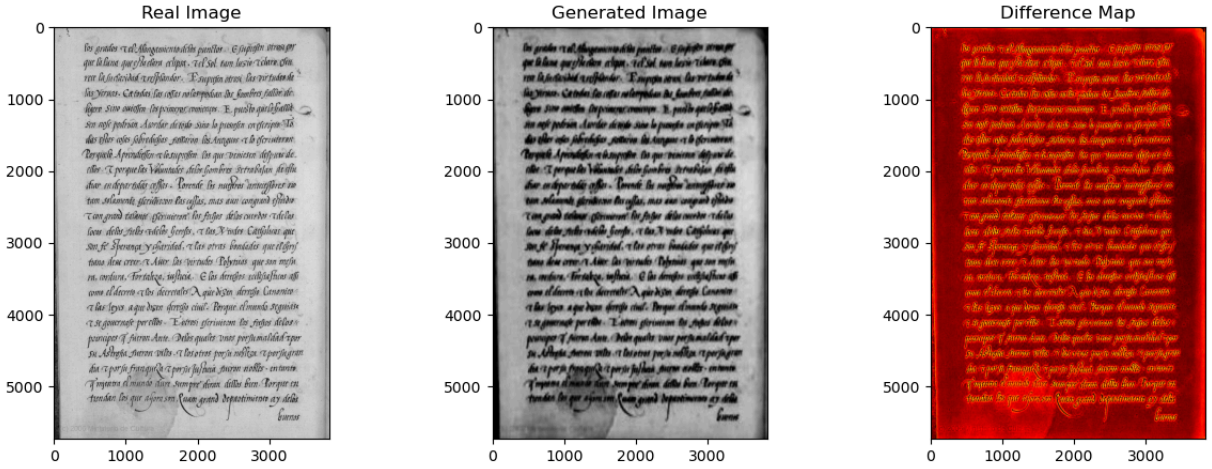
In [101…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/11.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/11.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



In [102…

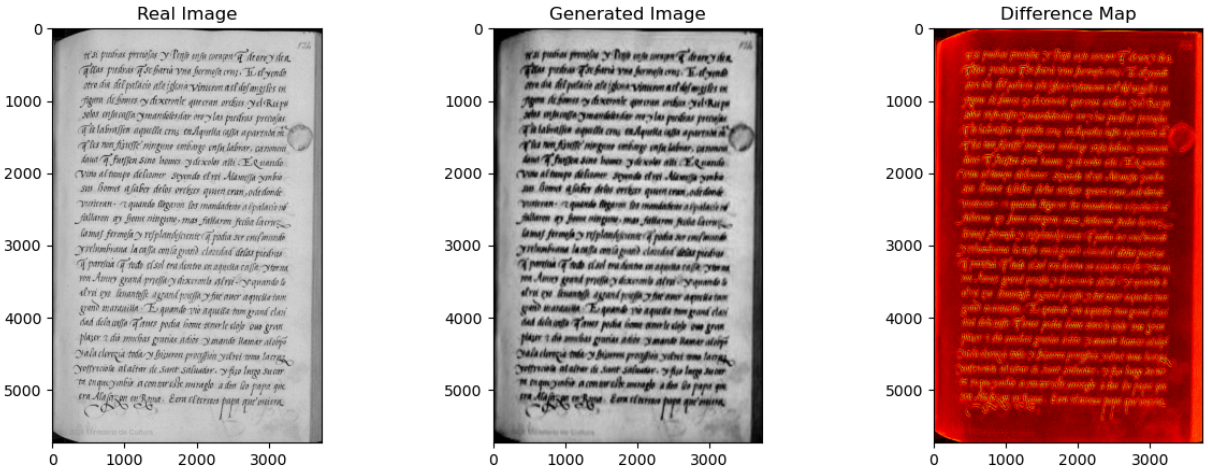
```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/274.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/274.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
```

```
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



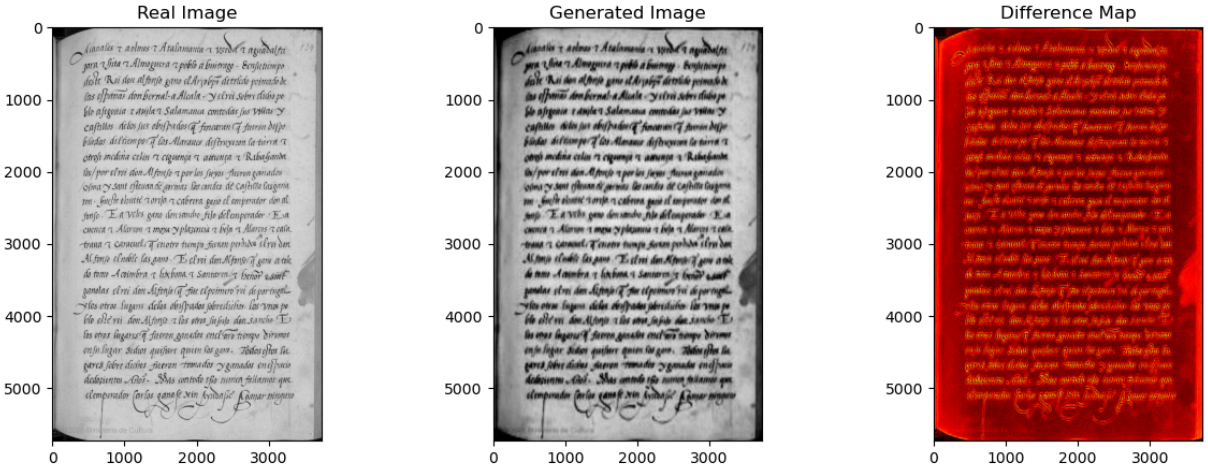
In [103…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/284.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/284.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



In [104…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/452.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/452.jpg"
```

```
real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```

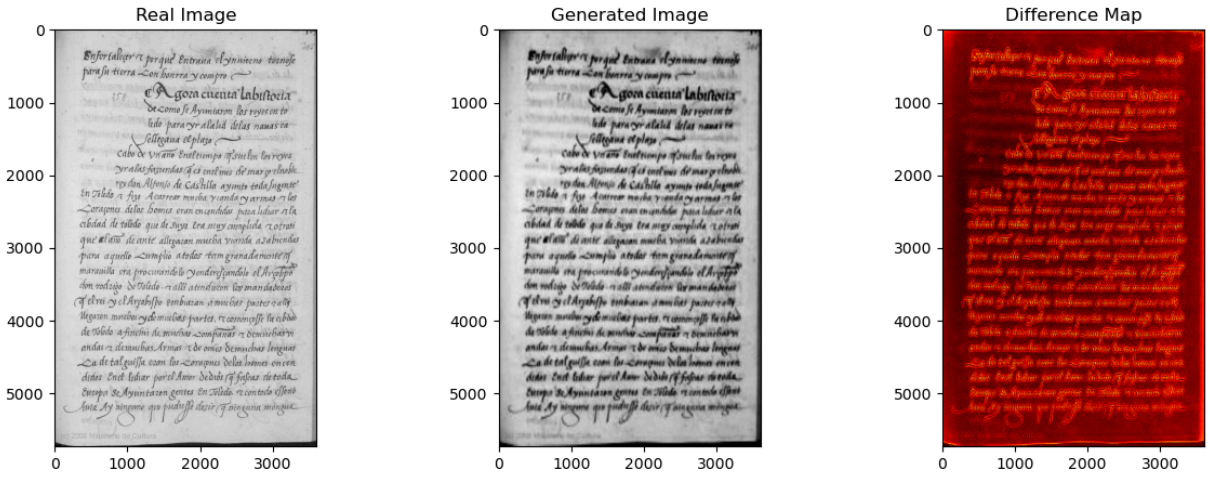
In [105…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/616.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/616.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



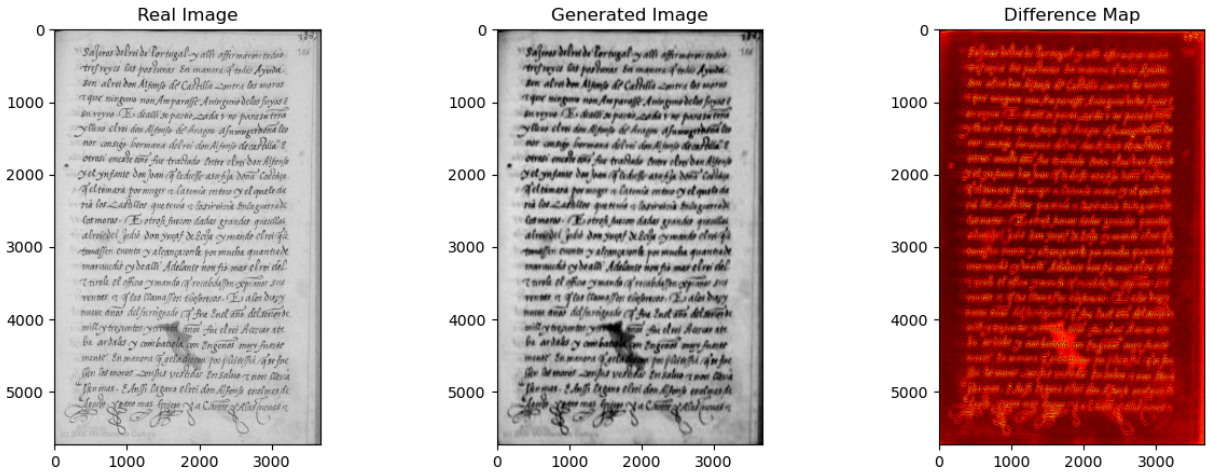
In [106…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/778.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/778.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



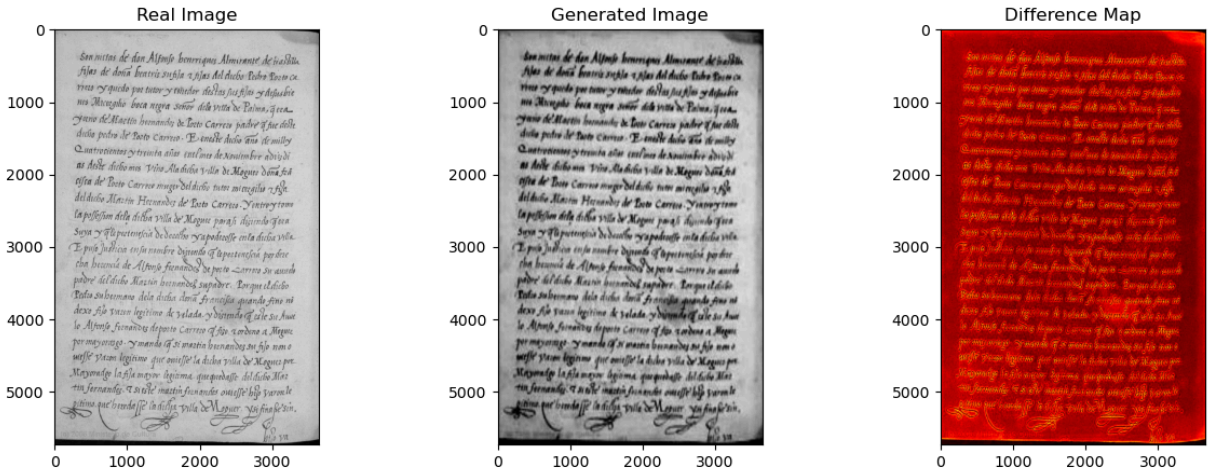
In [107…

```
# Image Comparisons
real_img_path = "/home/aniketj/GSOC_TASK3/TEST_IMAGES/845.jpg"
fake_img_path = "/home/aniketj/GSOC_TASK3/GENERATED_TEST_IMAGES/845.jpg"

real_img = cv2.imread(real_img_path, cv2.IMREAD_GRAYSCALE)
fake_img = cv2.imread(fake_img_path, cv2.IMREAD_GRAYSCALE)

fake_img = cv2.resize(fake_img, (real_img.shape[1], real_img.shape[0]))
```

```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(real_img, cmap='gray'); ax[0].set_title("Real Image")
ax[1].imshow(fake_img, cmap='gray'); ax[1].set_title("Generated Image")
diff = cv2.absdiff(real_img, fake_img)
ax[2].imshow(diff, cmap='hot'); ax[2].set_title("Difference Map")
plt.show()
```



In [ ]: