“GAN APPROACH FOR SKIN CANCER CLASSIFICATION”

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(Specialization in Artificial Intelligence & Machine Learning)

Submitted by

VU21CSEN0300096 LOHITHA PESALA
VU21CSEN0300038 P VENKAT SUJITHA
VU21CSEN0300019 KHYATHI LEKHNA U
VU21CSEN0300051 N SIVA RAMA KRISHNA

Under the esteemed guidance of
Dr. S S Nandini
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
VISAKHAPATNAM
2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)

DECLARATION

I hereby declare that the project report entitled **GAN APPROACH FOR SKIN CANCER CLASSIFICATION** is anoriginal work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering (AI&ML). The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 15-03-2025

Registration No(s)	Name(s)	Signature
VU21CSEN0300096	**LOHITHA PESALA**	
VU21CSEN0300038	**P VENKAT SUJITHA**	
VU21CSEN0300019	**U KHYATHI LEKHNA**	
VU21CSEN0300051	**N SIVA RAMA KRISHNA**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled **GAN APPROACH FOR SKIN CANCER CLASSIFICATION** is a bonafide record of work carried out by VU21CSEN0300096 - LOHITHA PESALA, VU21CSEN0300038 - P VENKAT SUJITHA, VU21CSEN0300019 - KHYATHI LEKHNA U, VU21CSEN0300051 – N SIVA RAMA KRISHNA students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering (AI&ML).

Date : 15-03-2025

Project Guide
Dr. S S Nandini

Head of the Department
Dr. Kuppili Naveen Kumar

ACKNOWLEDGEMENT

We are grateful to our HOD, Dr. K. Naveen Kumar, for his constant support and direction, which has motivated us to strive for excellence. His experience in AI and Machine Learning has given us a solid platform on which to construct our concept.

We would like to thank our in-class reviewers, Dr. Chandra Sekhar Potala, Assistant Professor for his crucial criticism and support during the duration of our research. Their intelligent ideas and constructive criticism really improved the quality of our work during the project.

We would like to thank our guide, Dr. S S Nandini, for her ongoing guidance and support during the project. Her wide knowledge, intelligent thoughts, and encouragement have been invaluable in guiding us through each aspect of the project. Although the project is still in progress, her patience and determination have given us the confidence and abilities we need to make consistent progress. We are appreciative for her willingness to offer her skills, and we look forward to completing this project with his important guidance.

Their collective support has been invaluable in leading us through the many stages of our project, and we look forward to implementing their suggestions as we near completion.

TABLE OF CONTENTS

1. Abstract

Skin cancer is a significant health risk due to its rapid progression, making early and accurate diagnosis essential. Traditional methods rely heavily on expert dermatologists, introducing subjectivity and limiting accessibility. Machine learning models, particularly those for image classification, hold promise for improving diagnostic accuracy but often struggle with limited, high-quality annotated skin lesion datasets necessary for robust training. This project mitigates data scarcity by using Deep Convolutional Generative Adversarial Networks (DCGANs) to generate synthetic skin lesion images, expanding dataset size and diversity. By replicating the complex patterns and textures of actual lesions, including rare cases, these synthetic images enhance training datasets. Combined with a fine-tuned VGG-19 Convolutional Neural Network (CNN) model, this dual approach improves classification accuracy between benign and malignant lesions. This GAN-based data augmentation and CNN classification method creates a more reliable diagnostic tool for skin cancer detection, ultimately assisting clinicians in providing better patient outcomes.

2. Introduction

Skin cancer is one of the most common cancers worldwide, and timely detection, especially of melanoma, is essential for improving survival rates. Dermatologists rely heavily on visual inspection of skin lesions for initial diagnoses, a process that can be subjective and error-prone, particularly in early stages. Machine learning and deep learning techniques have shown promise in automating skin cancer detection, potentially providing consistent and accurate diagnostic support. However, one major challenge limiting the effectiveness of machine learning in dermatology is the lack of large, annotated datasets of skin lesion images. Due to the rarity of certain lesions, privacy concerns, and labor-intensive labeling requirements, high-quality datasets in this domain remain scarce, leading to models with limited generalizability and reliability.

In recent years, GANs have emerged as powerful tools for data augmentation in medical imaging. Unlike traditional augmentation techniques, which involve simple transformations such as rotation or scaling, GANs can generate entirely new images that maintain the variability and detail of real images, making them highly suitable for expanding medical datasets. This project leverages the capabilities of GANs to generate synthetic skin lesion images, which, when combined with real images, create a more diverse and representative dataset. By training a CNN on this augmented dataset, the project aims to improve the classifier's ability to distinguish between benign and malignant lesions.

Specifically, this project employs a Deep Convolutional GAN (DCGAN) model to generate realistic, high-quality images of skin lesions. These images augment an existing dataset, and a VGG-19 model, pre-trained on a large dataset (e.g., ImageNet), is used as a classifier. Transfer learning enhances the classifier's performance on the augmented dataset, facilitating faster training and improved accuracy. This dual approach of synthetic data generation and advanced classification represents a significant advancement in the field of dermatology, potentially providing clinicians with a reliable tool to aid in early detection of skin cancer. This project's success could pave the way for wider applications of GANs in medical imaging, offering a sustainable solution to data scarcity and improving clinical outcomes in cancer diagnosis.

3. Literature Review

This section reviews studies and technologies used in skin lesion classification and data augmentation, focusing on GAN-based solutions and deep learning architectures for medical imaging.

[1] **GAN-Based Augmentation with Hybrid Loss for Dermoscopy Images**:

This study introduces a GAN model with a hybrid loss function to improve the quality of synthetic images for dermoscopy datasets. The hybrid loss combines adversarial loss, SSIM loss, and content loss, allowing the GAN to better capture both low- and high-frequency details. It demonstrates that GANs with carefully designed loss functions can improve data quality, which is critical for training CNNs in medical contexts (GAN).

[2] **Two-Stage Input-Space Image Augmentation and Interpretable Technique for Skin Cancer Diagnosis**:

This paper proposes a two-stage image augmentation method that generates new data for training classifiers. The approach enhances model accuracy and interpretability by focusing on explainable AI techniques. It shows how augmentation techniques can be combined with interpretability methods, providing clinicians with transparent and reliable diagnostic models for skin lesions (GAN).

[3] **Skin Lesion Analysis Using GANs: A Review**:

This review discusses various GAN applications in dermatology, particularly in generating synthetic skin lesion images for data augmentation. It highlights the strengths and limitations of different GAN architectures, including DCGAN and Pix2Pix, and provides insights into future research directions in using GANs for skin lesion analysis (GAN).

[4] **MelanoGANs: High-Resolution Skin Lesion Synthesis with GANs**:

MelanoGANs are designed for generating high-resolution images of skin lesions, specifically melanoma. This model uses a refined GAN architecture to capture complex lesion patterns and produce high-quality synthetic images. This approach addresses the need for detailed and realistic lesion images to improve classification accuracy when used in conjunction with CNNs (GAN).

4. Problem Identification & Objectives

4.1 Problem Identification

Early and accurate detection of skin cancer, particularly melanoma, is critical, as late-stage diagnoses significantly reduce survival rates. However, developing machine learning models for diagnosis faces challenges due to limited access to labeled skin lesion images, which are expensive and difficult to acquire. Basic augmentation techniques like flipping, rotating, and scaling have been used to expand datasets, but they fail to capture the true complexity of skin lesions, often leading to overfitting and poor generalization in real-world scenarios.

Generative Adversarial Networks (GANs) offer a promising solution by creating high-quality, realistic synthetic images that better represent diverse lesion characteristics. This approach improves dataset variety and model generalizability, enhancing diagnostic performance in skin cancer classification.

4.2 Objectives

- To develop a GAN to generate high-resolution synthetic skin lesion images, enhancing dataset size and diversity for more robust training.
- To build a CNN to classify skin lesions as benign or malignant, leveraging both real and GAN-generated images for improved accuracy and generalizability.
- To speed up training and boosting classification accuracy we used pre-trained VGG-19 network to initialize the CNN.
- Assessing model accuracy, sensitivity, specificity, and other metrics to ensure reliable diagnostic performance in clinical settings.
- To create a scalable, user-friendly system architecture ready for telemedicine integration and clinical deployment.

5. Existing System

Current approaches to automated skin cancer classification primarily rely on traditional data augmentation and CNN-based classification models. Data augmentation techniques, such as flipping, rotation, scaling, cropping, and color adjustments, add slight variations to existing images but fall short in introducing new lesion patterns or textures critical for diverse lesion classification. As a result, augmented datasets remain limited in representing real-world skin lesion diversity, leading to constrained generalizability in clinical applications.

CNN-based models, including ResNet, Inception, and VGG, have become standard for skin cancer classification, benefiting from their ability to extract spatial and hierarchical image features. Often, these models are pre-trained on large datasets like ImageNet and fine-tuned for specific tasks. However, even with traditional augmentation, these models can overfit limited datasets, achieving high accuracy on training data but struggling to generalize effectively to new, complex lesions. Transfer learning is also used, leveraging pre-trained models like VGG-19 or ResNet-50, but since these are based on non-medical images, they fail to address the lack of diverse skin lesion representations fully.

Attempts to introduce GAN-based augmentation, using models like DCGAN, Pix2Pix, and CycleGAN, are emerging but remain limited and experimental in skin cancer classification. GANs show potential for generating high-quality synthetic images, but challenges like mode collapse (limited diversity in output) and image artifacts reduce the clinical reliability of GAN-generated images. The scarcity of high-quality, annotated datasets, lack of meaningful diversity in traditional augmentation, and overfitting due to limited data all hinder current systems' performance and reliability in real-world applications.

Table 5.1: Summary of Limitations in Existing Systems

Component	**Description**	**Limitation**
Traditional Augmentation	Basic transformations like flipping, rotation, scaling	Limited diversity, fails to introduce new patterns
CNN Models	Convolutional networks like ResNet, VGG for classification	Overfitting due to data scarcity
Transfer Learning	Fine-tuning of pre-trained models on smaller datasets	Non-medical pre-training limits performance
GAN-based Attempts	Experimental use of GANs for generating synthetic lesion images	Mode collapse, image artifacts

In summary, while existing systems offer a starting point for skin cancer classification, they are insufficient for creating a robust diagnostic model. Traditional augmentation and transfer learning methods cannot overcome the data scarcity issue, and initial GAN implementations still struggle with quality and diversity. Thus, a more advanced GAN-based system that combines high-quality synthetic image generation with CNN-based classification could substantially improve performance and clinical applicability.

6. Proposed System

The Skin Lesion Synthesis with DCGAN project explores the generation of realistic images of skin lesions using a Deep Convolutional Generative Adversarial Network (DCGAN). This project aims to showcase the potential of DCGANs in generating synthetic images that closely resemble real skin lesions. Additionally, the project integrates the VGG-19 model for skin cancer classification, providing a comprehensive solution for both image synthesis and classification tasks.

Key Components

1. **DCGAN for Image Synthesis:**
    - The DCGAN architecture is employed to generate lifelike images of skin lesions.
    - The model is trained on a dataset compiled from Skincancer MNIST: HAM10000 hosted on kaggle.
    - The goal is to demonstrate that even with a small dataset, DCGANs can produce high-quality synthetic images useful for research and training.
2. **Data Collection:**
    - The primary dataset used for this project is available on Kaggle: Skin Cancer MNIST (HAM10000).
    - The dataset includes various skin lesion types, providing a comprehensive training set for the DCGAN.
3. **VGG-19 for Classification:**
    - The VGG-19 model, known for its effectiveness in image classification, is employed to classify skin cancer in the generated and real images.
    - The integration of classification enhances the utility of the project, allowing users to not only generate synthetic images but also assess their potential clinical relevance.
4. **GAN-VGG Pipeline:**
    - Feeds generated images into the VGG-19 classifier to assess the quality and utility of synthetic images.
    - Classifies both real and synthetic images, helping to evaluate the effectiveness of synthetic data in training.

7. System Architecture

Data Layer

- Data Source: Skincancer MNIST: HAM10000 dataset (contains labeled images of skin lesions for training).
- Data Preprocessing Module:
    - Data Augmentation: Increases dataset diversity by applying transformations like rotations, flips, and scaling.
    - Normalization: Normalizes pixel values to improve training stability for both the GAN and classifier.

Modeling Layer

- DCGAN Module:
    - Generator Network: Takes random noise as input and generates synthetic skin lesion images. Layers include:
        - Dense and reshape layers
        - Transposed convolutions
        - Batch normalization and ReLU activation
    - Discriminator Network: Takes both real and generated images, predicting if they are real or synthetic. Layers include:
        - Convolutional layers
        - Batch normalization and Leaky ReLU activation
        - Sigmoid activation for binary classification (real or fake).
    - Adversarial Training:
        - Loss Function: Binary cross-entropy loss for both Generator and Discriminator.
        - Training Loop: Alternates between training the Discriminator and Generator to optimize their respective objectives.
- VGG-19 Classifier Module:
    - Feature Extraction: Uses pre-trained VGG-19 to extract features from the image dataset.
    - Fine-Tuning: Adjusts layers for skin lesion classification with a custom output layer to classify skin lesion types.
    - Classification Head: Fully connected layers with softmax output for multi-class skin lesion classification.

Integration Layer

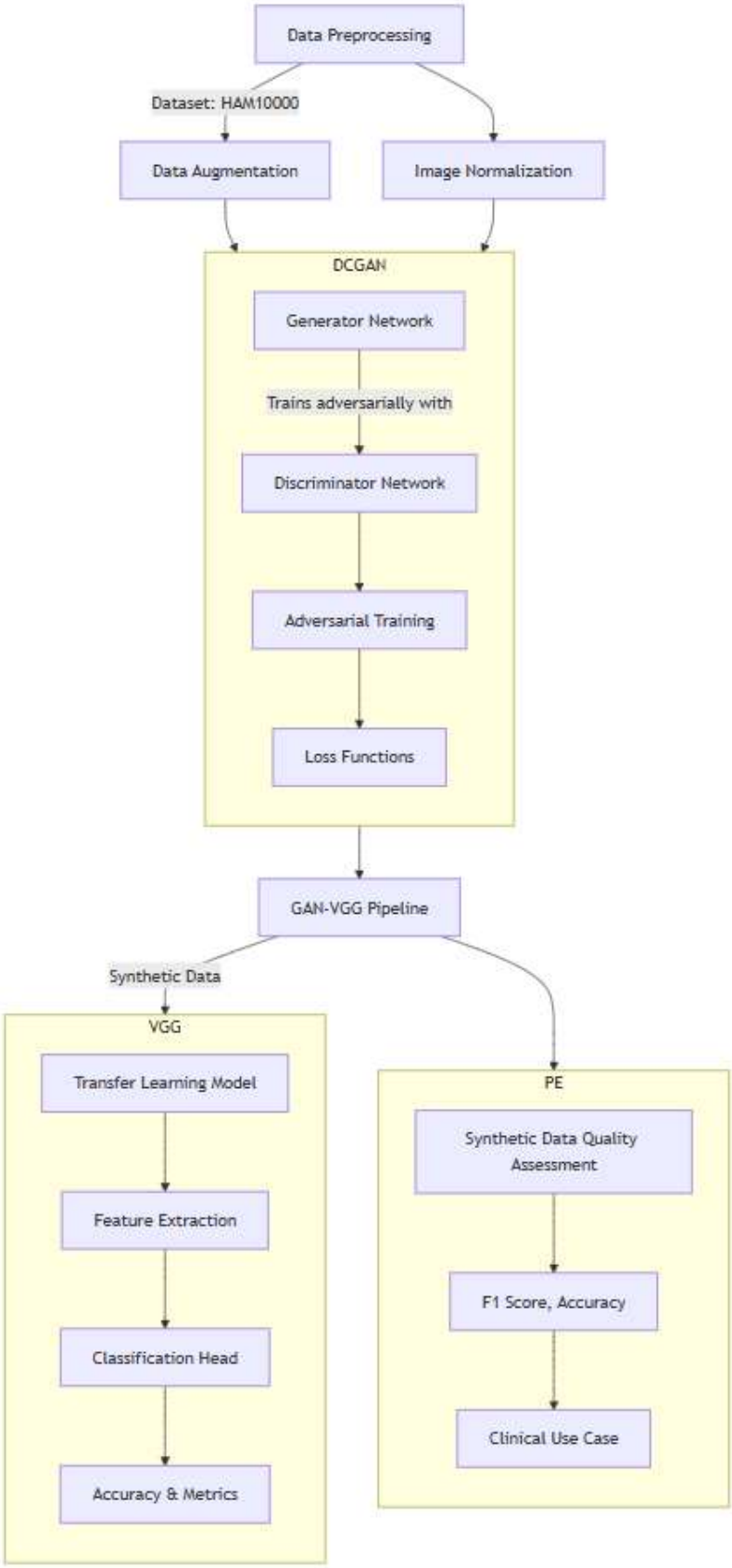
- GAN-VGG Pipeline:
    - Synthetic Data Validation: Feeds generated images into the VGG-19 classifier to assess the quality and utility of synthetic images.
    - Real & Synthetic Data Classification: Classifies both real and synthetic images, helping to evaluate the effectiveness of synthetic data in training.

Evaluation Layer

- Performance Metrics:
    - DCGAN Evaluation: Evaluates the Generator using metrics like the Inception Score (IS) or Fréchet Inception Distance (FID) to quantify the quality of synthetic images.
    - Classifier Evaluation: Evaluates the VGG-19 classifier using metrics like accuracy, F1 score, and loss to assess the quality of lesion classification.

Overall workflow

- Feed HAM10000 dataset to DCGAN.
- Store generated images in an array.
- Perform VGG-19 classification on HAM10000.
- Predict class of generated images.


Data Preprocessing
Dataset: HAM10000
Data Augmentation
Image Normalization
DCGAN
Generator Network
Trains adversarially with
Discriminator Network
Adversarial Training
Loss Functions
GAN-VGG Pipeline
Synthetic Data
VGG
Transfer Learning Model
Feature Extraction
Classification Head
Accuracy & Metrics
PE
Synthetic Data Quality Assessment
F1 Score, Accuracy
Clinical Use Case

Generator:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 8192)	827,392
leaky_re_lu_4 (LeakyReLU)	(None, 8192)	0
reshape (Reshape)	(None, 8, 8, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 128)	262,272
leaky_re_lu_5 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 128)	262,272
leaky_re_lu_6 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 128)	262,272
leaky_re_lu_7 (LeakyReLU)	(None, 64, 64, 128)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 128, 128, 128)	262,272
leaky_re_lu_8 (LeakyReLU)	(None, 128, 128, 128)	0
conv2d_4 (Conv2D)	(None, 128, 128, 3)	24,579

Total params: 1,901,059 (7.25 MB)
Trainable params: 1,901,059 (7.25 MB)
Non-trainable params: 0 (0.00 B)

Discriminator:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 128)	3,584
leaky_re_lu (LeakyReLU)	(None, 64, 64, 128)	0
conv2d_1 (Conv2D)	(None, 32, 32, 128)	147,584
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	147,584
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 8, 8, 128)	147,584
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dropout (Dropout)	(None, 8192)	0
dense (Dense)	(None, 1)	8,193

Total params: 454,529 (1.73 MB)
Trainable params: 454,529 (1.73 MB)
Non-trainable params: 0 (0.00 B)

8. Tools/Technologies Used

- Pandas and NumPy:
  - Libraries for data manipulation and preprocessing to organize and prepare images for model input.
- Scikit-Learn (sklearn):
  - Popular machine learning library, used for tasks like model selection, data splitting, and preprocessing.
- OpenCV:
  - Used for image processing tasks like resizing and normalization to ensure consistency in the dataset.
- TensorFlow/Keras:
  - Framework for building and training the DCGAN and VGG-19 models, facilitating the development of deep learning applications.
- Matplotlib and Seaborn:
  - Visualization libraries for analyzing training metrics such as accuracy and loss.
- PIL (Python Imaging Library):
  - Used to handle image processing tasks like loading, displaying, or manipulating images.
- Scipy:
  - Useful for scientific computations, such as statistics and numerical methods.
- Label Encoding and One-Hot Encoding (LabelEncoder, to_categorical):
  - Preprocessing steps to convert categorical labels into numerical form for machine learning algorithms.
- Optimizers and Activation Functions (Adam, LeakyReLU):
  - Specific functions in Keras used to optimize neural network training and control neuron activation behavior.
- Git:
  - Version control system for managing project files and tracking changes throughout development.

9. Implementation

Model 1: DCCGAN (Deep Convolutional Conditional GAN)

Architecture:

- A Deep Convolutional Conditional GAN (DCCGAN) is implemented to generate skin lesion images conditioned on class labels.
- The Generator is conditioned on both noise and class labels, enabling it to generate images corresponding to specific skin lesion types.
- The Discriminator takes both images and class labels as input, ensuring the model learns to differentiate between real and fake images based on class-specific features.
- The model architecture follows DCGAN principles but integrates label embeddings to improve class-specific generation.

Implementation:

1. **Preprocessing:**
    - Images are resized to 128×128 pixels.
    - One-hot encoding is used to represent class labels.
    - Data is normalized to the [-1,1] range for stable training.
2. **Model Training:**
    - The Generator takes both a random noise vector and class labels as input to generate conditioned images.
    - The Discriminator receives an image-label pair and classifies it as real or fake.
    - Both models are trained simultaneously using adversarial training.
    - Loss function: Binary Cross-Entropy (BCE) with Adam optimizer.

Model 2: DCGAN (Deep Convolutional GAN) Model

Architecture:

- A Deep Convolutional Generative Adversarial Network (DCGAN) is implemented to generate realistic skin lesion images.
- The Generator takes a latent noise vector and class label as input and generates synthetic images using transposed convolutional layers.
- The Discriminator classifies images as real or fake using convolutional layers with batch normalization and LeakyReLU activations.
- The model is trained adversarially using the binary cross-entropy loss function and Adam optimizer.

Implementation:

1. **Preprocessing:**
    - Images are resized to 128×128 pixels.
    - Normalization is applied to scale pixel values between -1 and 1.
    - Data augmentation is performed to increase diversity and prevent overfitting.
2. **Model Training:**
    - The Generator takes a random noise vector and class embedding to generate synthetic images.
    - The Discriminator receives both real and synthetic images and classifies them accordingly.
    - The models are trained adversarially with alternating updates.
    - Loss function: Binary Cross-Entropy (BCE) with Adam optimizer.

Model 3: VGG-19 for Skin Cancer Classification

Architecture

- Pre-trained VGG-19 model used as a feature extractor.
- Additional CNN layers fine-tune high-level representations.
- Fully connected layers map extracted features to class probabilities.
- Softmax activation for multi-class classification.

Implementation

1. **Preprocessing**
    - Images resized to 128×128 pixels.
    - Normalization applied to match pre-trained model input.
2. **Training**
    - VGG-19 layers frozen to retain learned features.
    - Additional layers trained using categorical cross-entropy loss.
    - Adam optimizer for efficient gradient updates.

Advantages

- Faster convergence using pre-trained weights.
- Better feature extraction than traditional CNNs.

10. Results

1. DCGAN Performance

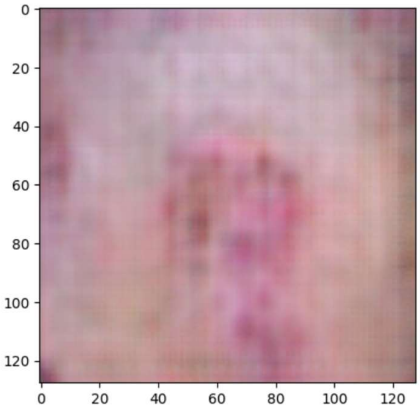
- Discriminator Loss (DCGAN): -0.8025
- Generator Loss (DCGAN): 0.2375
- Observation: The discriminator is performing strongly, suggesting it effectively distinguishes real from synthetic images. However, the generator's low loss may indicate mode collapse, meaning it might be generating less diverse images.

2. DCGAN Performance

- Discriminator Loss: -0.2813
- Generator Loss: 0.5982
- Comparison: The discriminator has become more confident, but the generator may need tuning to improve diversity.

3. VGG-19 Classifier Performance

- Accuracy on Real & Synthetic Skin Lesion Images: 69%
- Observation: The classifier is moderately effective but may struggle with fine-grained differences. Further augmentation and training could help.



Generated image using DCGAN

	precision	recall	f1-score	support
bcc	0.76	0.72	0.74	152
bkl	0.64	0.60	0.62	216
mel	0.67	0.73	0.70	218
accuracy			0.68	586
macro avg	0.69	0.69	0.69	586
weighted avg	0.68	0.68	0.68	586

Performance metrics of VGG model (considering 3 classes)

9. Conclusion

In conclusion, this project presents an innovative approach to skin cancer classification by integrating Deep Convolutional GANs (DCGAN) for data augmentation and VGG-19 for image classification. By generating high-quality synthetic skin lesion images, the project expands the dataset with diverse samples, addressing the limitations of traditional medical image datasets, which are often small and imbalanced. This data enrichment strategy enhances the model's ability to generalize and improve classification accuracy between benign and malignant lesions, demonstrating the effectiveness of combining GANs with CNNs for improved diagnostic reliability.

The success of DCGAN and VGG-19 in augmenting and classifying skin lesions underscores the transformative role of GANs in medical imaging, particularly in fields where data availability is limited. This hybrid approach not only strengthens the model's predictive capability but also offers a foundation for AI-driven diagnostic tools, enhancing early detection rates for skin cancer and paving the way for accessible, AI-assisted healthcare solutions.

10. References

[1] E. Goceri, "GAN based augmentation using a hybrid loss function for dermoscopy images," *GAN based augmentation using a hybrid loss function for dermoscopy images,* vol. 1, no. 2, p. 6, 2024.

[2] A. S. J. Z. a. A. W. Catur Supriyanto, “Two-Stage Input-Space Image Augmentation and Interpretable Technique for Accurate and Explainable Skin Cancer Diagnosis,” vol. I, no. 1, p. 14, 2023.

[3] S. Q. G. Oge Marques, "Skin lesion analysis using generative adversarial networks: a review," vol. II, no. 1, p. 43, 2023.

[4] S. Albarqouni, "MelanoGANs: High Resolution Skin Lesion Synthesis with GANs," vol. I, no. 1, p. 9, 2018.

[5] J. Z. Z. L. Y. Z. Y. C. X. X. ,. J. L. ,. J. X. Saisai Ding, "High-resolution dermoscopy image synthesis with conditional generative adversarial networks," *Biomedical Signal Processing and Control,* vol. 64, no. 102224, p. 5, 2021.

11. Annexure 1 (Source Code)

```
# Functionality
import os
from glob import glob
from google.colab import files
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sys
assert sys.version_info >= (3, 5)
import sklearn
assert sklearn.__version__ >= "0.20"

try:
    IS_COLAB = True
except Exception:
    IS_COLAB = False

# TensorFlow ≥2.0 is required
import tensorflow as tf
from tensorflow import keras
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from keras.models import load_model
from numpy.random import randn

from PIL import Image
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected.")

!pip install -q kaggle
```

```
!mkdir -p ~/.kaggle

files.upload()

!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d kmader/skin-cancer-mnist-ham10000

!unzip skin-cancer-mnist-ham10000.zip

"""Data pre-processing"""

base_skin_dir = "/content/"

df_meta = pd.read_csv(os.path.join(base_skin_dir,'HAM10000_metadata.csv'))
df_meta.head()

# Functionality so that we can find the image path for each metadata entry
imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(base_skin_dir, '*',
'*.jpg'))}

lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}

tile_df = pd.read_csv(os.path.join(base_skin_dir,
'HAM10000_metadata.csv'))
tile_df['path'] = tile_df['image_id'].map(imageid_path_dict.get)
tile_df['cell_type'] = tile_df['dx'].map(lesion_type_dict.get)
tile_df['cell_type_idx'] = pd.Categorical(tile_df['cell_type']).codes
tile_df.sample(3)
tile_df.describe(exclude=[np.number])
fig, ax1 = plt.subplots(1, 1, figsize = (10, 5))
tile_df['cell_type'].value_counts().plot(kind='bar', ax=ax1)
categories = ['mel', 'bkl', 'bcc']
tile_df = tile_df[tile_df.dx.isin(categories)]

from skimage.io import imread
tile_df['image'] = tile_df['path'].map(imread)
print(tile_df)

# Image size distributions
tile_df['image'].map(lambda x: x.shape).value_counts()
```

```
n_samples = 5
fig, m_axs = plt.subplots(3, n_samples, figsize = (4*n_samples, 3*3))
for n_axs, (type_name, type_rows) in zip(m_axs,
                                         tile_df.sort_values(['cell_type']
).groupby('cell_type')):
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples,
random_state=2018).iterrows()):
        c_ax.imshow(c_row['image'])
        c_ax.axis('off')
fig.savefig('category_samples.png', dpi=300)

np.random.seed(42)
skin_df = pd.read_csv('/content/HAM10000_metadata.csv')

SIZE=64*2
le = LabelEncoder()
le.fit(skin_df['dx'])
LabelEncoder()
print(list(le.classes_))
skin_df['label'] = le.transform(skin_df["dx"])
print(skin_df.sample(10))
fig = plt.figure(figsize=(15,10))
ax1 = fig.add_subplot(221)
skin_df['dx'].value_counts().plot(kind='bar', ax=ax1)
ax1.set_ylabel('Count')
ax1.set_title('Cell Type');
ax2 = fig.add_subplot(222)
skin_df['sex'].value_counts().plot(kind='bar', ax=ax2)
ax2.set_ylabel('Count', size=15)
ax2.set_title('Sex');
ax3 = fig.add_subplot(223)
skin_df['localization'].value_counts().plot(kind='bar')
ax3.set_ylabel('Count',size=12)
ax3.set_title('Localization')
ax4 = fig.add_subplot(224)
sample_age = skin_df[pd.notnull(skin_df['age'])]
sns.distplot(sample_age['age'], fit=stats.norm, color='red')
ax4.set_title('Age')
plt.tight_layout()
plt.show()


"""GAN"""

# df_0 = skin_df[skin_df['label'] == 0]
df_1 = skin_df[skin_df['label'] == 1] #Only BCC
# df_2 = skin_df[skin_df['label'] == 2]
```

```
# df_3 = skin_df[skin_df['label'] == 3]
# df_4 = skin_df[skin_df['label'] == 4]
# df_5 = skin_df[skin_df['label'] == 5]
# df_6 = skin_df[skin_df['label'] == 6]

skin_df_balanced = pd.concat([ df_1])
print(skin_df_balanced['label'].value_counts())

image_path = {os.path.splitext(os.path.basename(x))[0]: x
                    for x in glob(os.path.join('/content/', '*',
'*.jpg'))}
skin_df_balanced['path'] = skin_df['image_id'].map(image_path.get)
skin_df_balanced['image'] = skin_df_balanced['path'].map(lambda x:
np.asarray(Image.open(x).resize((SIZE,SIZE))))
print(skin_df_balanced['path'])

import tensorflow as tf
from tensorflow import keras
from keras.utils import to_categorical
# from keras.utils.np_utils import to_categorical # used for converting
labels to one-hot-encoding
X = np.asarray(skin_df_balanced['image'].tolist())
Y=skin_df_balanced['label']     #Assign label values to Y
print(Y)
Y_cat = to_categorical(Y, num_classes=2) #Convert to categorical as this
is a multiclass classification problem

trainX, testX, trainy, testy = train_test_split(X, Y_cat, test_size=0.3,
random_state=42)
train_merged = np.concatenate((trainX, testX))
display(train_merged)

for i in Y:
  if (i == 0):
    arr = X[i]
    print(skin_df_balanced['path'][i])
    break

plt.matshow(arr)
plt.savefig("image0.png")

for i in range(25):
  plt.subplot(5, 5, 1 + i)
  plt.axis('off')
  plt.imshow(trainX[i])
plt.show()

def define_discriminator(in_shape=(128,128,3)):
  model = Sequential()
```

```
  model.add(Conv2D(128, (3,3), strides=(2,2),
padding='same',input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(learning_rate=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
  return model
test_discr = define_discriminator()

print(test_discr.summary())

def define_generator(latent_dim):
  model = Sequential()

  n_nodes = 128 * 8 * 8  #8192 nodes
  model.add(Dense(n_nodes, input_dim=latent_dim)) #Dense layer so we can
work with 1D latent vector
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((8, 8, 128)))  #8x8x128 dataset from the latent
vector.
  # upsample to 16x16
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
#16x16x128
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 32x32
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
#32x32x128
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))#64*64*128
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
  model.add(LeakyReLU(alpha=0.2))#64*64*128

  # generate
  model.add(Conv2D(3, (8,8), activation='tanh', padding='same')) #32x32x3
  return model  #Model not compiled as it is not directly trained like the
discriminator.
                    #Generator is trained via GAN combined model.
```

```
test_gen = define_generator(100)

print(test_gen.summary())

def define_gan(generator, discriminator):
  discriminator.trainable = False  #Discriminator is trained separately.
So set to not trainable.
  # connect generator and discriminator
  model = Sequential()
  model.add(generator)
  model.add(discriminator)
  # compile model
  opt = Adam(learning_rate=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt)
  return model

def load_real_samples():

  X = train_merged.astype('float32')
  # scale from [0,255] to [-1,1]
  X = (X - 127.5) / 127.5  #Generator uses tanh activation so rescale
                           #original images to -1 to 1 to match the output
of generator.
  # print(X)
  return X

def generate_real_samples(dataset, n_samples):
  # choose random images
  ix = randint(0, dataset.shape[0], n_samples)
  # select the random images and assign it to X
  X = dataset[ix]
  # generate class labels and assign to y
  y = ones((n_samples, 1)) #Label=1 indicating they are real
  return X, y

def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

def generate_fake_samples(generator, latent_dim, n_samples):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n_samples)
  # predict using generator to generate fake samples.
  X = generator.predict(x_input)
  # Class labels will be 0 as these samples are fake.
```

```
  y = zeros((n_samples, 1))  #Label=0 indicating they are fake
  return X, y

def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=1000,
n_batch=128):
  bat_per_epo = int(dataset.shape[0] / n_batch)
  half_batch = int(n_batch / 2)  #the discriminator model is updated for a
half batch of real samples
                            #and a half batch of fake samples, combined a
single batch.
  # manually enumerate epochs and bacthes.
  for i in range(n_epochs):
    # enumerate batches over the training set
    for j in range(bat_per_epo):

            # Train the discriminator on real and fake images, separately
(half batch each)
        #Research showed that separate training is more effective.
      # get randomly selected 'real' samples
      X_real, y_real = generate_real_samples(dataset, half_batch)
      # update discriminator model weights
            ##train_on_batch allows you to update weights based on a
collection
            #of samples you provide
            #Let us just capture loss and ignore accuracy value (2nd
output below)
      d_loss_real, _ = d_model.train_on_batch(X_real, y_real)

            # generate 'fake' examples
      X_fake, y_fake = generate_fake_samples(g_model, latent_dim,
half_batch)
      # update discriminator model weights
      d_loss_fake, _ = d_model.train_on_batch(X_fake, y_fake)

      # prepare points in latent space as input for the generator
      X_gan = generate_latent_points(latent_dim, n_batch)

            # The generator wants the discriminator to label the generated
samples
        # as valid (ones)
        #This is where the generator is trying to trick discriminator into
believing
        #the generated image is true (hence value of 1 for y)
      y_gan = ones((n_batch, 1))

            # Generator is part of combined model where it got directly
linked with the discriminator
        # Train the generator with latent_dim as x and 1 as y.
```

```
        # Again, 1 as the output as it is adversarial and if generator did
a great
        #job of folling the discriminator then the output would be 1
(true)
      # update the generator via the discriminator's error
      g_loss = gan_model.train_on_batch(X_gan, y_gan)

            # Print losses on this batch
      print('Epoch>%d, Batch %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
        (i+1, j+1, bat_per_epo, d_loss_real, d_loss_fake, g_loss))
  # save the generator model
  g_model.save('GAN_best.keras')

latent_dim = 100
discriminator = define_discriminator()
generator = define_generator(latent_dim)
gan_model = define_gan(generator, discriminator)
dataset = load_real_samples()
train(generator, discriminator, gan_model, dataset, latent_dim)

from keras.models import load_model
model = load_model('GAN_best.keras')
num_iterations = 8
num_samples = 25
A = []
for _ in range(num_iterations):
    latent_points = generate_latent_points(100, num_samples)
    X = model.predict(latent_points)
    # Scale from [-1,1] to [0,1]
    X = (X + 1) / 2.0
    X = (X * 255).astype(np.uint8)
    A.append(X)

A = np.array(A)
A = A.reshape(200,128,128,3)
print(A.shape)
plt.matshow(A[0])
for i in range(200):
  plt.subplot(20, 20, 1 + i)
  plt.axis('off')
  plt.imshow(A[i])
plt.show()
plt.imshow(A[72])

skin_df = pd.read_csv('/content/HAM10000_metadata.csv')
SIZE=128
# label encoding to numeric values from text
le = LabelEncoder()
le.fit(skin_df['dx'])
```

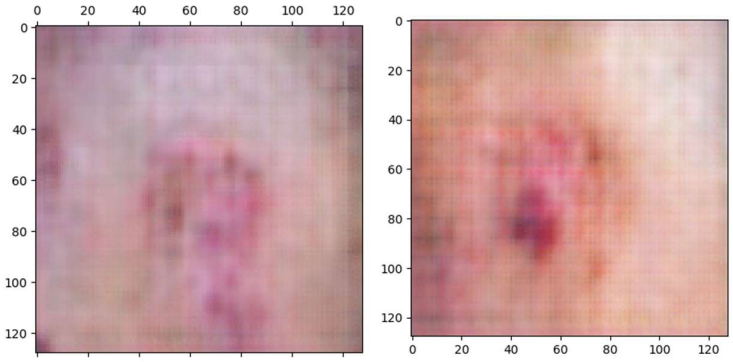
```
LabelEncoder()
print(list(le.classes_))
skin_df['label'] = le.transform(skin_df["dx"])
# print(skin_df.sample(10))

df_1 = skin_df[skin_df['label'] == 1] #bcc
df_2 = skin_df[skin_df['label'] == 2] #bkl
df_4 = skin_df[skin_df['label'] == 4] #mel
skin_df_balanced = pd.concat([ df_1, df_2, df_4])
skin_df_balanced['label'] = skin_df_balanced['label'].replace(1, 0)
skin_df_balanced['label'] = skin_df_balanced['label'].replace(2, 1)
skin_df_balanced['label'] = skin_df_balanced['label'].replace(4, 2)

image_path = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join('/content/', '*',
'*.jpg'))}
#Define the path and add as a new column
skin_df_balanced['path'] = skin_df['image_id'].map(image_path.get)
#Use the path to read images.
skin_df_balanced['image'] = skin_df_balanced['path'].map(lambda x:
np.asarray(Image.open(x).resize((SIZE,SIZE))))
X = np.asarray(skin_df_balanced['image'].tolist())
Y=skin_df_balanced['label'] #Assign label values to Y

B = [0] * (A.shape[0])
B=np.array(B)
B = pd.DataFrame(B)
print(B.shape)
print(B)

frames = [B, Y]
Yf = pd.concat(frames)
print(Yf)

Xf=np.concatenate((A,X))
Xf.shape
Yf = to_categorical(Yf, num_classes=3)
print(Xf.shape,Yf.shape)

x_train_auto, x_test_auto, y_train_auto, y_test_auto =
train_test_split(Xf, Yf, test_size=0.2, random_state=42)
print(x_train_auto.shape,y_train_auto.shape)
# print(y_train_auto)

vgg=tf.keras.applications.VGG19(input_shape=(128,128,3),weights='imagenet'
, include_top=False)
for i in vgg.layers:
    i.trainable = False
```

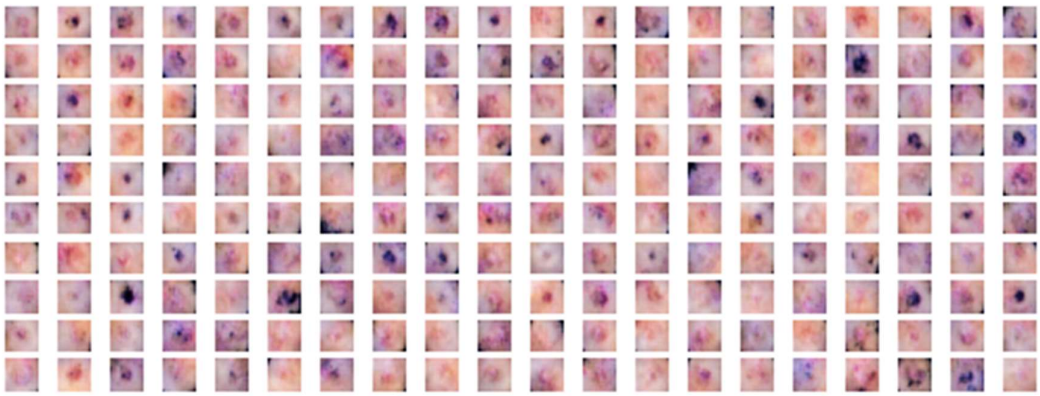
```
model = tf.keras.models.Sequential([
    vgg,
    tf.keras.layers.Conv2D(64, (3, 3), padding="same"),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(32, (3, 3), padding="same"),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(3, activation = 'softmax')
])
print(model.summary())

callback = tf.keras.callbacks.ModelCheckpoint(filepath='/content/',
                      monitor='val_acc',
                      mode='max',verbose=1)

model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam',
               metrics = ['accuracy'])
history = model.fit(x_train_auto,
                    y_train_auto,
                    validation_split=0.2,
                    batch_size = 64,
                    epochs = 100,
                    )#callbacks=[callback]

model.save("VGG.h5")

new_model=keras.models.load_model("VGG.h5")

new_model.summary()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

```
plt.show()

model.save('vgg19_best.keras')
model= tf.keras.models.load_model('vgg19_best.keras')
loss, acc = model.evaluate(x_test_auto, y_test_auto, verbose=2)

import sklearn.metrics as metrics
y_pred_ohe = model.predict(x_test_auto)
y_pred_ohe = np.argmax(y_pred_ohe,axis=1)
y_pred_labels = np.argmax(y_test_auto, axis=1)
confusion_matrix = metrics.confusion_matrix(y_true=y_pred_labels,
y_pred=y_pred_ohe)
class_labels = ['bcc','bkl','mel']
report = sklearn.metrics.classification_report(y_pred_labels, y_pred_ohe,
target_names = class_labels)
print(report)

cm = sklearn.metrics.confusion_matrix(y_pred_labels, y_pred_ohe)
plt.figure(figsize=(8,8))
sns.heatmap(cm, fmt='.0f', cmap="crest", annot=True, linewidths=0.2 )
plt.title('confusion matrix')
plt.xlabel('predicted value')
plt.ylabel('Truth value')
plt.show()
print(sklearn.metrics.confusion_matrix(y_pred_labels, y_pred_ohe))

A[0].shape
img = np.expand_dims(A[0], axis=0)
plt.imshow(A[0])
plt.show
predictions = model.predict(img)
output = np.argmax(predictions, axis=1)

model.save_weights("VGG_weights.h5")
model.load_weights("VGG_weights.h5")

def process_image(image_path):
    img = Image.open(image_path)
    img = img.resize((128, 128))  # Resize the image to the input size
expected by your CNN model
    #img = np.expand_dims(img, axis=0)
    img = np.asarray(img)
    return img

processed=process_image('/content/HAM10000_images_part_1/ISIC_0024306.jpg'
)
processed.shape
plt.matshow(processed)
```

13. Annexure 2 (Output)

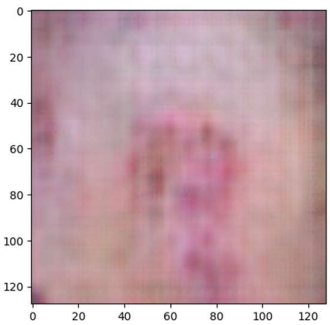
Generated Images using DCGAN





VGG Classifier

	precision	recall	f1-score	support
bcc	0.76	0.72	0.74	152
bkl	0.64	0.60	0.62	216
mel	0.67	0.73	0.70	218
accuracy			0.68	586
macro avg	0.69	0.69	0.69	586
weighted avg	0.68	0.68	0.68	586



Prediction

```
output = np.argmax(predictions, axis=1)
output
```

```
array([0])
```