```python
import os
from PIL import Image
import matplotlib.pyplot as plt

from keras.layers import Input, Dense, Reshape, Flatten
from keras.layers import Activation
from glob import glob
from keras.models import Sequential, Model
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import sys
import numpy as np

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"
print("Dataset path:", dataset_path)
```

```
Dataset path: /content/drive/MyDrive/HEART US IMAGE FETUS
```

```python
from glob import glob

dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"
print("Dataset path:", dataset_path)

# Use glob to get all image paths in the specified directory
image_paths = glob(dataset_path + '/*.jpg')

# Print all image paths
print("Image paths:")
for path in image_paths:
    print(path)
```

```
Dataset path: /content/drive/MyDrive/HEART US IMAGE FETUS
Image paths:
/content/drive/MyDrive/HEART US IMAGE FETUS/1950612.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/531643.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1830719.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947161.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1761038.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1741778.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184842.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184845.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/531639.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/878716.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1741782.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/910907.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1829682.jpg
```

```
/content/drive/MyDrive/HEART US IMAGE FETUS/1828656.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947162.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1829684.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1766431.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1766429.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184841.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1829680.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1830127.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1158827.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/531640.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1829253.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184843.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/531642.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875066.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950611.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1806269.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931522.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882730.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184844.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1830718.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1829679.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1795969.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184902.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/910911.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931523.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875070.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/531630.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902291.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1891555.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875062.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1158799.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875058.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875055.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882868.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875059.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1891560.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875054.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1886834.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1766430.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882866.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1886839.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1828928.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875052.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875051.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1886825.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184900.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1886840.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875063.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889650.jpg
```

```
/content/drive/MyDrive/HEART US IMAGE FETUS/1931524.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875130.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/877955.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902289.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902305.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889676.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902304.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902288.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/877956.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875128.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/878717.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1830717.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1886838.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882731.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184838.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875127.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912631.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912591.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875060.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1908509.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917262.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1910702.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875065.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917254.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1908510.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1887391.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902292.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889678.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1741780.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889685.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889645.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1875069.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1910701.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882739.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917255.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912627.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1882740.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1891562.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1889682.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902293.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902287.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912628.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1936137.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917257.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917268.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1910761.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917328.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1184901.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1910734.jpg
```

```
/content/drive/MyDrive/HEART US IMAGE FETUS/1912590.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917253.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1936139.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917267.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1923234.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917265.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1923230.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940641.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912624.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931521.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931529.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917327.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912626.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940653.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902286.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1923231.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931532.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912605.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1912630.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1936136.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945545.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1910763.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1917261.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945600.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1902301.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940640.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940663.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931525.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1923233.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940642.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940701.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1936135.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1929103.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940728.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940647.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945552.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945551.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946146.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1931495.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946158.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946147.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945713.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946161.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946157.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940636.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946159.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946000.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1939941.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946347.jpg
```

```
/content/drive/MyDrive/HEART US IMAGE FETUS/1946374.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945549.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940643.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946149.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940645.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945983.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940644.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946156.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946150.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945986.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946155.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946211.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945982.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940633.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946653.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947159.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946148.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945712.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946373.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945987.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945989.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946349.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947150.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1945546.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946022.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946645.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947151.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1939986.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947155.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947163.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946353.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949793.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947156.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947164.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946354.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946002.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949583.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949407.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946365.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949439.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949584.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946344.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946021.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946222.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949647.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947152.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949394.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946643.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1940649.jpg
```

```
/content/drive/MyDrive/HEART US IMAGE FETUS/1946367.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947153.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949775.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949589.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949597.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949945.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950148.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950552.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949794.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949942.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947158.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949648.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947160.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1946345.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949972.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950188.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949406.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947000.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947165.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949471.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950150.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950151.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949400.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950152.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950185.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949588.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950187.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950556.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949452.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950549.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1951010.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1951151.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950554.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950564.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949879.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949469.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950560.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950184.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950551.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949943.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949592.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1947157.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950610.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949435.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1949963.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950831.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950550.jpg
/content/drive/MyDrive/HEART US IMAGE FETUS/1950586.jpg
```

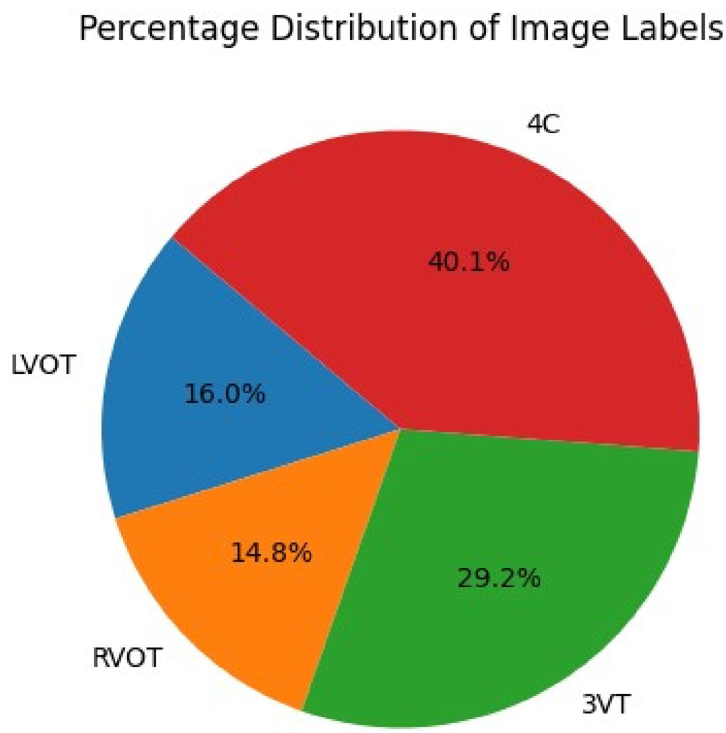
```
# Count the number of images in the dataset
original_size = len(image_paths)

# Print the original size of the dataset
print("Original size of the dataset:", original_size)
```

```
Original size of the dataset: 257
```

```
import os
import pandas as pd
import matplotlib.pyplot as plt

dataset_path = '/content/drive/MyDrive/HEART US IMAGE FETUS'

# Get a list of all image files in the dataset folder
image_files = [file for file in os.listdir(dataset_path) if
file.endswith(('.jpg', '.png', '.jpeg'))]

# List of image labels
image_labels = ['LVOT', 'RVOT',  '3VT', '4C']

# List of corresponding counts
part_counts = [41,38,75,103]

# Calculate the total number of images
total_images = sum(part_counts)

# Calculate the percentage distribution of each label
percentage_distribution = [count / total_images * 100 for count in
part_counts]

# Create a DataFrame with filenames, labels, and counts
label_list = [f"{count}-{label}" for count, label in zip(part_counts,
image_labels) for _ in range(count)]
df = pd.DataFrame({'Image': image_files[:len(label_list)], 'Label':
label_list})

# Display the percentage distribution
for label, percentage in zip(image_labels, percentage_distribution):
    print(f"{label}: {percentage:.2f}%")

# Create a pie chart
plt.figure(figsize=(5, 5))
plt.pie(percentage_distribution, labels=image_labels, autopct='%1.1f%
%', startangle=140)
plt.title('Percentage Distribution of Image Labels')
plt.show()
```

```
LVOT: 15.95%
RVOT: 14.79%
```

```
3VT: 29.18%
4C: 40.08%
```




```
from PIL import Image
import os

dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"

def check_image_color_mode(image_path):
    with Image.open(image_path) as img:
        return img.mode

def check_dataset_color_mode(dataset_path):
    for filename in os.listdir(dataset_path):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(dataset_path, filename)
            color_mode = check_image_color_mode(image_path)
            print(f"{filename}: {color_mode}")

# Call the function to check color modes in the dataset
check_dataset_color_mode(dataset_path)

1950612.jpg: RGB
531643.jpg: RGB
```

```
1830719.jpg: RGB
1947161.jpg: RGB
1761038.jpg: RGB
1741778.jpg: RGB
1184842.jpg: RGB
1184845.jpg: RGB
531639.jpg: RGB
878716.jpg: RGB
1741782.jpg: RGB
910907.jpg: RGB
1829682.jpg: RGB
1828656.jpg: RGB
1947162.jpg: RGB
1829684.jpg: RGB
1766431.jpg: RGB
1766429.jpg: RGB
1184841.jpg: RGB
1829680.jpg: RGB
1830127.jpg: RGB
1158827.jpg: RGB
531640.jpg: RGB
1829253.jpg: RGB
1184843.jpg: RGB
531642.jpg: RGB
1875066.jpg: RGB
1950611.jpg: RGB
1806269.jpg: RGB
1931522.jpg: RGB
1882730.jpg: RGB
1184844.jpg: RGB
1830718.jpg: RGB
1829679.jpg: RGB
1795969.jpg: RGB
1184902.jpg: RGB
910911.jpg: RGB
1931523.jpg: RGB
1875070.jpg: RGB
531630.jpg: RGB
1902291.jpg: RGB
1891555.jpg: RGB
1875062.jpg: RGB
1158799.jpg: RGB
1875058.jpg: RGB
1875055.jpg: RGB
1882868.jpg: RGB
1875059.jpg: RGB
1891560.jpg: RGB
1875054.jpg: RGB
1886834.jpg: RGB
```

```
1766430.jpg: RGB
1882866.jpg: RGB
1886839.jpg: RGB
1828928.jpg: RGB
1875052.jpg: RGB
1875051.jpg: RGB
1886825.jpg: RGB
1184900.jpg: RGB
1886840.jpg: RGB
1875063.jpg: RGB
1889650.jpg: RGB
1931524.jpg: RGB
1875130.jpg: RGB
877955.jpg: RGB
1902289.jpg: RGB
1902305.jpg: RGB
1889676.jpg: RGB
1902304.jpg: RGB
1902288.jpg: RGB
877956.jpg: RGB
1875128.jpg: RGB
878717.jpg: RGB
1830717.jpg: RGB
1886838.jpg: RGB
1882731.jpg: RGB
1184838.jpg: RGB
1875127.jpg: RGB
1912631.jpg: RGB
1912591.jpg: RGB
1875060.jpg: RGB
1908509.jpg: RGB
1917262.jpg: RGB
1910702.jpg: RGB
1875065.jpg: RGB
1917254.jpg: RGB
1908510.jpg: RGB
1887391.jpg: RGB
1902292.jpg: RGB
1889678.jpg: RGB
1741780.jpg: RGB
1889685.jpg: RGB
1889645.jpg: RGB
1875069.jpg: RGB
1910701.jpg: RGB
1882739.jpg: RGB
1917255.jpg: RGB
1912627.jpg: RGB
1882740.jpg: RGB
1891562.jpg: RGB
```

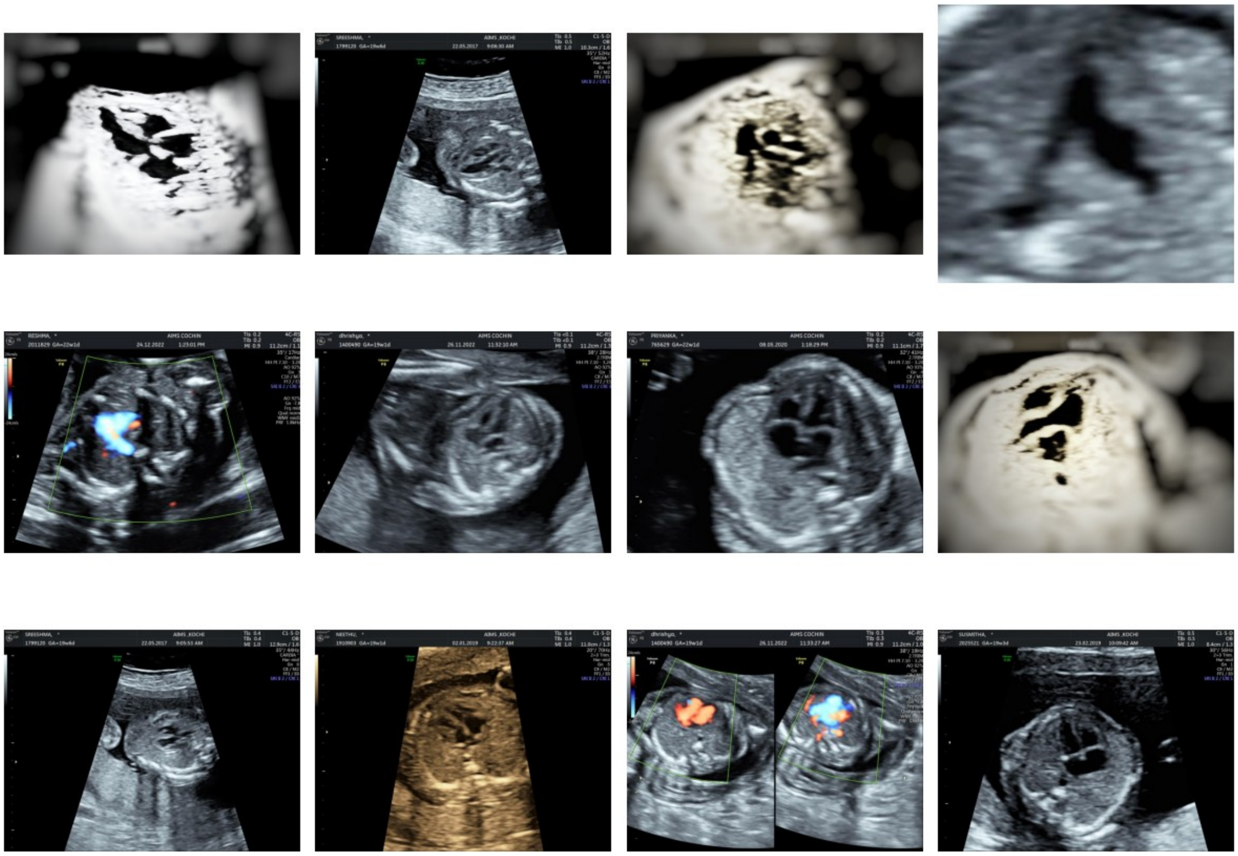
```
1889682.jpg: RGB
1902293.jpg: RGB
1902287.jpg: RGB
1912628.jpg: RGB
1936137.jpg: RGB
1917257.jpg: RGB
1917268.jpg: RGB
1910761.jpg: RGB
1917328.jpg: RGB
1184901.jpg: RGB
1910734.jpg: RGB
1912590.jpg: RGB
1917253.jpg: RGB
1936139.jpg: RGB
1917267.jpg: RGB
1923234.jpg: RGB
1917265.jpg: RGB
1923230.jpg: RGB
1940641.jpg: RGB
1912624.jpg: RGB
1931521.jpg: RGB
1931529.jpg: RGB
1917327.jpg: RGB
1912626.jpg: RGB
1940653.jpg: RGB
1902286.jpg: RGB
1923231.jpg: RGB
1931532.jpg: RGB
1912605.jpg: RGB
1912630.jpg: RGB
1936136.jpg: RGB
1945545.jpg: RGB
1910763.jpg: RGB
1917261.jpg: RGB
1945600.jpg: RGB
1902301.jpg: RGB
1940640.jpg: RGB
1940663.jpg: RGB
1931525.jpg: RGB
1923233.jpg: RGB
1940642.jpg: RGB
1940701.jpg: RGB
1936135.jpg: RGB
1929103.jpg: RGB
1940728.jpg: RGB
1940647.jpg: RGB
1945552.jpg: RGB
1945551.jpg: RGB
1946146.jpg: RGB
```

```
1931495.jpg: RGB
1946158.jpg: RGB
1946147.jpg: RGB
1945713.jpg: RGB
1946161.jpg: RGB
1946157.jpg: RGB
1940636.jpg: RGB
1946159.jpg: RGB
1946000.jpg: RGB
1939941.jpg: RGB
1946347.jpg: RGB
1946374.jpg: RGB
1945549.jpg: RGB
1940643.jpg: RGB
1946149.jpg: RGB
1940645.jpg: RGB
1945983.jpg: RGB
1940644.jpg: RGB
1946156.jpg: RGB
1946150.jpg: RGB
1945986.jpg: RGB
1946155.jpg: RGB
1946211.jpg: RGB
1945982.jpg: RGB
1940633.jpg: RGB
1946653.jpg: RGB
1947159.jpg: RGB
1946148.jpg: RGB
1945712.jpg: RGB
1946373.jpg: RGB
1945987.jpg: RGB
1945989.jpg: RGB
1946349.jpg: RGB
1947150.jpg: RGB
1945546.jpg: RGB
1946022.jpg: RGB
1946645.jpg: RGB
1947151.jpg: RGB
1939986.jpg: RGB
1947155.jpg: RGB
1947163.jpg: RGB
1946353.jpg: RGB
1949793.jpg: RGB
1947156.jpg: RGB
1947164.jpg: RGB
1946354.jpg: RGB
1946002.jpg: RGB
1949583.jpg: RGB
1949407.jpg: RGB
```

```
1946365.jpg: RGB
1949439.jpg: RGB
1949584.jpg: RGB
1946344.jpg: RGB
1946021.jpg: RGB
1946222.jpg: RGB
1949647.jpg: RGB
1947152.jpg: RGB
1949394.jpg: RGB
1946643.jpg: RGB
1940649.jpg: RGB
1946367.jpg: RGB
1947153.jpg: RGB
1949775.jpg: RGB
1949589.jpg: RGB
1949597.jpg: RGB
1949945.jpg: RGB
1950148.jpg: RGB
1950552.jpg: RGB
1949794.jpg: RGB
1949942.jpg: RGB
1947158.jpg: RGB
1949648.jpg: RGB
1947160.jpg: RGB
1946345.jpg: RGB
1949972.jpg: RGB
1950188.jpg: RGB
1949406.jpg: RGB
1947000.jpg: RGB
1947165.jpg: RGB
1949471.jpg: RGB
1950150.jpg: RGB
1950151.jpg: RGB
1949400.jpg: RGB
1950152.jpg: RGB
1950185.jpg: RGB
1949588.jpg: RGB
1950187.jpg: RGB
1950556.jpg: RGB
1949452.jpg: RGB
1950549.jpg: RGB
1951010.jpg: RGB
1951151.jpg: RGB
1950554.jpg: RGB
1950564.jpg: RGB
1949879.jpg: RGB
1949469.jpg: RGB
1950560.jpg: RGB
1950184.jpg: RGB
```

```
1950551.jpg: RGB
1949943.jpg: RGB
1949592.jpg: RGB
1947157.jpg: RGB
1950610.jpg: RGB
1949435.jpg: RGB
1949963.jpg: RGB
1950831.jpg: RGB
1950550.jpg: RGB
1950586.jpg: RGB
```

```
import matplotlib.image as mpimg
import os
num_images_to_display = 12

# List all files in the dataset directory
files = os.listdir(dataset_path)

# Select the first 12 images
selected_images = files[:num_images_to_display]

# Create a subplot with rows and columns
rows = 3  # Change this based on your preference
columns = 4  # Change this based on your preference
fig, axs = plt.subplots(rows, columns, figsize=(12, 9))

# Display each image in the subplot
for i in range(rows):
    for j in range(columns):
        img_path = os.path.join(dataset_path, selected_images[i *
columns + j])
        img = mpimg.imread(img_path)
        axs[i, j].imshow(img, cmap='gray')
        axs[i, j].axis('off')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```



```python
from PIL import Image
import os

dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"

def check_image_properties(image_path):
    with Image.open(image_path) as img:
        return img.size, img.mode

def check_dataset_properties(dataset_path):
    for filename in os.listdir(dataset_path):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(dataset_path, filename)
            size, color_mode = check_image_properties(image_path)
            print(f"{filename}: Size={size}, Mode={color_mode}")

# Call the function to check properties of images in the dataset
check_dataset_properties(dataset_path)
```

```
1950612.jpg: Size=(800, 600), Mode=RGB
531643.jpg: Size=(1136, 852), Mode=RGB
1830719.jpg: Size=(1136, 852), Mode=RGB
1947161.jpg: Size=(258, 243), Mode=RGB
1761038.jpg: Size=(970, 727), Mode=RGB
```

```
1741778.jpg: Size=(970, 727), Mode=RGB
1184842.jpg: Size=(970, 727), Mode=RGB
1184845.jpg: Size=(970, 727), Mode=RGB
531639.jpg: Size=(1136, 852), Mode=RGB
878716.jpg: Size=(1136, 852), Mode=RGB
1741782.jpg: Size=(970, 727), Mode=RGB
910907.jpg: Size=(1136, 852), Mode=RGB
1829682.jpg: Size=(970, 727), Mode=RGB
1828656.jpg: Size=(1136, 852), Mode=RGB
1947162.jpg: Size=(800, 600), Mode=RGB
1829684.jpg: Size=(970, 727), Mode=RGB
1766431.jpg: Size=(1136, 852), Mode=RGB
1766429.jpg: Size=(1136, 852), Mode=RGB
1184841.jpg: Size=(970, 727), Mode=RGB
1829680.jpg: Size=(970, 727), Mode=RGB
1830127.jpg: Size=(1136, 852), Mode=RGB
1158827.jpg: Size=(800, 564), Mode=RGB
531640.jpg: Size=(1136, 852), Mode=RGB
1829253.jpg: Size=(970, 727), Mode=RGB
1184843.jpg: Size=(387, 356), Mode=RGB
531642.jpg: Size=(1136, 852), Mode=RGB
1875066.jpg: Size=(1136, 852), Mode=RGB
1950611.jpg: Size=(435, 305), Mode=RGB
1806269.jpg: Size=(1136, 852), Mode=RGB
1931522.jpg: Size=(305, 228), Mode=RGB
1882730.jpg: Size=(970, 727), Mode=RGB
1184844.jpg: Size=(970, 727), Mode=RGB
1830718.jpg: Size=(487, 378), Mode=RGB
1829679.jpg: Size=(970, 727), Mode=RGB
1795969.jpg: Size=(970, 727), Mode=RGB
1184902.jpg: Size=(970, 727), Mode=RGB
910911.jpg: Size=(1136, 852), Mode=RGB
1931523.jpg: Size=(800, 600), Mode=RGB
1875070.jpg: Size=(1136, 852), Mode=RGB
531630.jpg: Size=(1136, 852), Mode=RGB
1902291.jpg: Size=(800, 600), Mode=RGB
1891555.jpg: Size=(800, 600), Mode=RGB
1875062.jpg: Size=(1136, 852), Mode=RGB
1158799.jpg: Size=(800, 564), Mode=RGB
1875058.jpg: Size=(1136, 852), Mode=RGB
1875055.jpg: Size=(1136, 852), Mode=RGB
1882868.jpg: Size=(970, 727), Mode=RGB
1875059.jpg: Size=(1136, 852), Mode=RGB
1891560.jpg: Size=(800, 600), Mode=RGB
1875054.jpg: Size=(1136, 852), Mode=RGB
1886834.jpg: Size=(970, 727), Mode=RGB
1766430.jpg: Size=(1136, 852), Mode=RGB
1882866.jpg: Size=(970, 727), Mode=RGB
1886839.jpg: Size=(970, 727), Mode=RGB
```

```
1828928.jpg: Size=(1136, 852), Mode=RGB
1875052.jpg: Size=(1136, 852), Mode=RGB
1875051.jpg: Size=(1136, 852), Mode=RGB
1886825.jpg: Size=(970, 727), Mode=RGB
1184900.jpg: Size=(970, 727), Mode=RGB
1886840.jpg: Size=(970, 727), Mode=RGB
1875063.jpg: Size=(1136, 852), Mode=RGB
1889650.jpg: Size=(800, 600), Mode=RGB
1931524.jpg: Size=(800, 600), Mode=RGB
1875130.jpg: Size=(1136, 852), Mode=RGB
877955.jpg: Size=(800, 600), Mode=RGB
1902289.jpg: Size=(800, 600), Mode=RGB
1902305.jpg: Size=(800, 600), Mode=RGB
1889676.jpg: Size=(800, 600), Mode=RGB
1902304.jpg: Size=(800, 600), Mode=RGB
1902288.jpg: Size=(800, 600), Mode=RGB
877956.jpg: Size=(800, 600), Mode=RGB
1875128.jpg: Size=(1136, 852), Mode=RGB
878717.jpg: Size=(1136, 852), Mode=RGB
1830717.jpg: Size=(1136, 852), Mode=RGB
1886838.jpg: Size=(970, 727), Mode=RGB
1882731.jpg: Size=(970, 727), Mode=RGB
1184838.jpg: Size=(970, 727), Mode=RGB
1875127.jpg: Size=(1136, 852), Mode=RGB
1912631.jpg: Size=(800, 600), Mode=RGB
1912591.jpg: Size=(800, 600), Mode=RGB
1875060.jpg: Size=(1136, 852), Mode=RGB
1908509.jpg: Size=(800, 600), Mode=RGB
1917262.jpg: Size=(800, 600), Mode=RGB
1910702.jpg: Size=(800, 600), Mode=RGB
1875065.jpg: Size=(1136, 852), Mode=RGB
1917254.jpg: Size=(800, 600), Mode=RGB
1908510.jpg: Size=(800, 600), Mode=RGB
1887391.jpg: Size=(970, 727), Mode=RGB
1902292.jpg: Size=(800, 600), Mode=RGB
1889678.jpg: Size=(800, 600), Mode=RGB
1741780.jpg: Size=(970, 727), Mode=RGB
1889685.jpg: Size=(800, 600), Mode=RGB
1889645.jpg: Size=(800, 600), Mode=RGB
1875069.jpg: Size=(1136, 852), Mode=RGB
1910701.jpg: Size=(800, 600), Mode=RGB
1882739.jpg: Size=(970, 727), Mode=RGB
1917255.jpg: Size=(800, 600), Mode=RGB
1912627.jpg: Size=(800, 600), Mode=RGB
1882740.jpg: Size=(970, 727), Mode=RGB
1891562.jpg: Size=(800, 600), Mode=RGB
1889682.jpg: Size=(800, 600), Mode=RGB
1902293.jpg: Size=(800, 600), Mode=RGB
1902287.jpg: Size=(800, 600), Mode=RGB
```

```
1912628.jpg: Size=(800, 600), Mode=RGB
1936137.jpg: Size=(800, 600), Mode=RGB
1917257.jpg: Size=(800, 600), Mode=RGB
1917268.jpg: Size=(800, 600), Mode=RGB
1910761.jpg: Size=(800, 600), Mode=RGB
1917328.jpg: Size=(800, 600), Mode=RGB
1184901.jpg: Size=(970, 727), Mode=RGB
1910734.jpg: Size=(800, 600), Mode=RGB
1912590.jpg: Size=(800, 600), Mode=RGB
1917253.jpg: Size=(800, 600), Mode=RGB
1936139.jpg: Size=(800, 600), Mode=RGB
1917267.jpg: Size=(800, 600), Mode=RGB
1923234.jpg: Size=(800, 600), Mode=RGB
1917265.jpg: Size=(800, 600), Mode=RGB
1923230.jpg: Size=(800, 600), Mode=RGB
1940641.jpg: Size=(800, 600), Mode=RGB
1912624.jpg: Size=(800, 600), Mode=RGB
1931521.jpg: Size=(800, 600), Mode=RGB
1931529.jpg: Size=(800, 600), Mode=RGB
1917327.jpg: Size=(800, 600), Mode=RGB
1912626.jpg: Size=(800, 600), Mode=RGB
1940653.jpg: Size=(800, 600), Mode=RGB
1902286.jpg: Size=(800, 600), Mode=RGB
1923231.jpg: Size=(800, 600), Mode=RGB
1931532.jpg: Size=(800, 600), Mode=RGB
1912605.jpg: Size=(800, 600), Mode=RGB
1912630.jpg: Size=(800, 600), Mode=RGB
1936136.jpg: Size=(800, 600), Mode=RGB
1945545.jpg: Size=(800, 600), Mode=RGB
1910763.jpg: Size=(800, 600), Mode=RGB
1917261.jpg: Size=(800, 600), Mode=RGB
1945600.jpg: Size=(800, 600), Mode=RGB
1902301.jpg: Size=(800, 600), Mode=RGB
1940640.jpg: Size=(800, 600), Mode=RGB
1940663.jpg: Size=(800, 600), Mode=RGB
1931525.jpg: Size=(800, 600), Mode=RGB
1923233.jpg: Size=(800, 600), Mode=RGB
1940642.jpg: Size=(800, 600), Mode=RGB
1940701.jpg: Size=(800, 600), Mode=RGB
1936135.jpg: Size=(800, 600), Mode=RGB
1929103.jpg: Size=(800, 600), Mode=RGB
1940728.jpg: Size=(800, 600), Mode=RGB
1940647.jpg: Size=(800, 600), Mode=RGB
1945552.jpg: Size=(800, 600), Mode=RGB
1945551.jpg: Size=(800, 600), Mode=RGB
1946146.jpg: Size=(800, 600), Mode=RGB
1931495.jpg: Size=(800, 600), Mode=RGB
1946158.jpg: Size=(800, 600), Mode=RGB
1946147.jpg: Size=(800, 600), Mode=RGB
```

```
1945713.jpg: Size=(800, 600), Mode=RGB
1946161.jpg: Size=(800, 600), Mode=RGB
1946157.jpg: Size=(800, 600), Mode=RGB
1940636.jpg: Size=(800, 600), Mode=RGB
1946159.jpg: Size=(800, 600), Mode=RGB
1946000.jpg: Size=(800, 600), Mode=RGB
1939941.jpg: Size=(800, 600), Mode=RGB
1946347.jpg: Size=(800, 600), Mode=RGB
1946374.jpg: Size=(800, 600), Mode=RGB
1945549.jpg: Size=(800, 600), Mode=RGB
1940643.jpg: Size=(800, 600), Mode=RGB
1946149.jpg: Size=(800, 600), Mode=RGB
1940645.jpg: Size=(800, 600), Mode=RGB
1945983.jpg: Size=(800, 600), Mode=RGB
1940644.jpg: Size=(800, 600), Mode=RGB
1946156.jpg: Size=(800, 600), Mode=RGB
1946150.jpg: Size=(800, 600), Mode=RGB
1945986.jpg: Size=(800, 600), Mode=RGB
1946155.jpg: Size=(800, 600), Mode=RGB
1946211.jpg: Size=(800, 600), Mode=RGB
1945982.jpg: Size=(800, 600), Mode=RGB
1940633.jpg: Size=(800, 600), Mode=RGB
1946653.jpg: Size=(800, 600), Mode=RGB
1947159.jpg: Size=(800, 600), Mode=RGB
1946148.jpg: Size=(800, 600), Mode=RGB
1945712.jpg: Size=(800, 600), Mode=RGB
1946373.jpg: Size=(800, 600), Mode=RGB
1945987.jpg: Size=(800, 600), Mode=RGB
1945989.jpg: Size=(800, 600), Mode=RGB
1946349.jpg: Size=(800, 600), Mode=RGB
1947150.jpg: Size=(800, 600), Mode=RGB
1945546.jpg: Size=(800, 600), Mode=RGB
1946022.jpg: Size=(800, 600), Mode=RGB
1946645.jpg: Size=(800, 600), Mode=RGB
1947151.jpg: Size=(800, 600), Mode=RGB
1939986.jpg: Size=(800, 600), Mode=RGB
1947155.jpg: Size=(800, 600), Mode=RGB
1947163.jpg: Size=(800, 600), Mode=RGB
1946353.jpg: Size=(800, 600), Mode=RGB
1949793.jpg: Size=(800, 600), Mode=RGB
1947156.jpg: Size=(800, 600), Mode=RGB
1947164.jpg: Size=(800, 600), Mode=RGB
1946354.jpg: Size=(800, 600), Mode=RGB
1946002.jpg: Size=(800, 600), Mode=RGB
1949583.jpg: Size=(800, 600), Mode=RGB
1949407.jpg: Size=(800, 600), Mode=RGB
1946365.jpg: Size=(800, 600), Mode=RGB
1949439.jpg: Size=(800, 600), Mode=RGB
1949584.jpg: Size=(800, 600), Mode=RGB
```

```
1946344.jpg: Size=(800, 600), Mode=RGB
1946021.jpg: Size=(800, 600), Mode=RGB
1946222.jpg: Size=(800, 600), Mode=RGB
1949647.jpg: Size=(800, 600), Mode=RGB
1947152.jpg: Size=(800, 600), Mode=RGB
1949394.jpg: Size=(800, 600), Mode=RGB
1946643.jpg: Size=(800, 600), Mode=RGB
1940649.jpg: Size=(800, 600), Mode=RGB
1946367.jpg: Size=(800, 600), Mode=RGB
1947153.jpg: Size=(800, 600), Mode=RGB
1949775.jpg: Size=(800, 600), Mode=RGB
1949589.jpg: Size=(800, 600), Mode=RGB
1949597.jpg: Size=(800, 600), Mode=RGB
1949945.jpg: Size=(800, 600), Mode=RGB
1950148.jpg: Size=(800, 600), Mode=RGB
1950552.jpg: Size=(800, 600), Mode=RGB
1949794.jpg: Size=(800, 600), Mode=RGB
1949942.jpg: Size=(800, 600), Mode=RGB
1947158.jpg: Size=(800, 600), Mode=RGB
1949648.jpg: Size=(800, 600), Mode=RGB
1947160.jpg: Size=(800, 600), Mode=RGB
1946345.jpg: Size=(800, 600), Mode=RGB
1949972.jpg: Size=(800, 600), Mode=RGB
1950188.jpg: Size=(800, 600), Mode=RGB
1949406.jpg: Size=(800, 600), Mode=RGB
1947000.jpg: Size=(800, 600), Mode=RGB
1947165.jpg: Size=(800, 600), Mode=RGB
1949471.jpg: Size=(800, 600), Mode=RGB
1950150.jpg: Size=(800, 600), Mode=RGB
1950151.jpg: Size=(800, 600), Mode=RGB
1949400.jpg: Size=(800, 600), Mode=RGB
1950152.jpg: Size=(800, 600), Mode=RGB
1950185.jpg: Size=(800, 600), Mode=RGB
1949588.jpg: Size=(800, 600), Mode=RGB
1950187.jpg: Size=(800, 600), Mode=RGB
1950556.jpg: Size=(800, 600), Mode=RGB
1949452.jpg: Size=(800, 600), Mode=RGB
1950549.jpg: Size=(800, 600), Mode=RGB
1951010.jpg: Size=(800, 600), Mode=RGB
1951151.jpg: Size=(800, 600), Mode=RGB
1950554.jpg: Size=(800, 600), Mode=RGB
1950564.jpg: Size=(800, 600), Mode=RGB
1949879.jpg: Size=(800, 600), Mode=RGB
1949469.jpg: Size=(800, 600), Mode=RGB
1950560.jpg: Size=(800, 600), Mode=RGB
1950184.jpg: Size=(800, 600), Mode=RGB
1950551.jpg: Size=(800, 600), Mode=RGB
1949943.jpg: Size=(800, 600), Mode=RGB
1949592.jpg: Size=(800, 600), Mode=RGB
```

```
1947157.jpg: Size=(800, 600), Mode=RGB
1950610.jpg: Size=(800, 600), Mode=RGB
1949435.jpg: Size=(800, 600), Mode=RGB
1949963.jpg: Size=(800, 600), Mode=RGB
1950831.jpg: Size=(800, 600), Mode=RGB
1950550.jpg: Size=(800, 600), Mode=RGB
1950586.jpg: Size=(800, 600), Mode=RGB
```

PREPROCESSING

A) RESIZING

```
from PIL import Image
import os

# Path to the original dataset
original_dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"

# Path to the new location to store resized images
resized_dataset_path =
"/content/drive/MyDrive/Resized_HEART_US_IMAGE_FETUS"

# Create the new directory if it doesn't exist
if not os.path.exists(resized_dataset_path):
    os.makedirs(resized_dataset_path)

# Function to resize and save images
def resize_and_save(image_path, output_path, target_size=(256, 256)):
    with Image.open(image_path) as img:
        resized_img = img.resize(target_size, Image.ANTIALIAS)
        resized_img.save(output_path)

# Loop through each image in the original dataset
for filename in os.listdir(original_dataset_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        # Create the full path for the original and resized images
        original_image_path = os.path.join(original_dataset_path,
filename)
        resized_image_path = os.path.join(resized_dataset_path,
filename)

        # Resize and save the image
        resize_and_save(original_image_path, resized_image_path)

# Display some resized images for reference
num_images_to_display = 4
selected_resized_images = os.listdir(resized_dataset_path)
[:num_images_to_display]
```

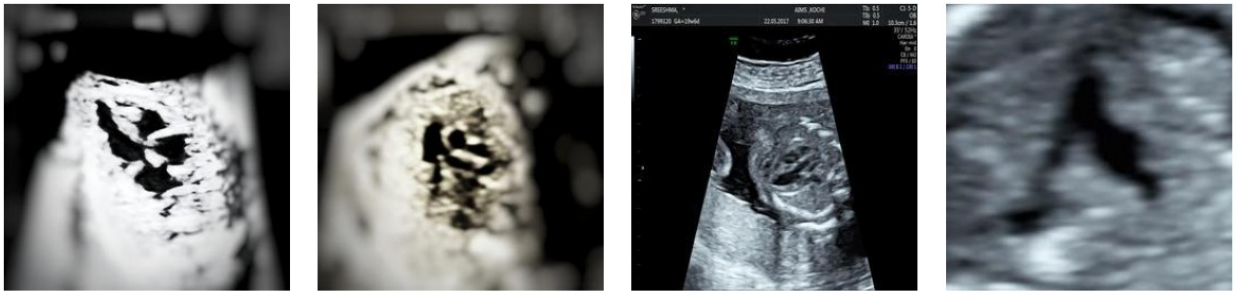
```python
# Create a subplot with rows and columns
rows = 1
columns = num_images_to_display
fig, axs = plt.subplots(rows, columns, figsize=(12, 3))

# Display each resized image in the subplot
for i in range(columns):
    img_path = os.path.join(resized_dataset_path,
selected_resized_images[i])
    img = mpimg.imread(img_path)
    axs[i].imshow(img, cmap='gray')
    axs[i].axis('off')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```

```
<ipython-input-12-746f01e1b3a1>:17: DeprecationWarning: ANTIALIAS is
deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS
or Resampling.LANCZOS instead.
  resized_img = img.resize(target_size, Image.ANTIALIAS)
```

NORMALIZATION

```python
import numpy as np

# Path to the resized dataset
resized_dataset_path =
"/content/drive/MyDrive/Resized_HEART_US_IMAGE_FETUS"

# Path to the location to store normalized images
normalized_dataset_path =
"/content/drive/MyDrive/Normalized_HEART_US_IMAGE_FETUS"

# Create the new directory if it doesn't exist
if not os.path.exists(normalized_dataset_path):
    os.makedirs(normalized_dataset_path)

# Function to normalize and save images
```

```
def normalize_and_save(image_path, output_path):
    with Image.open(image_path) as img:
        # Convert image to NumPy array
        img_array = np.array(img)

        # Normalize pixel values to the range [0, 1]
        normalized_img = img_array / 255.0

        # Save the normalized image
        normalized_img = Image.fromarray((normalized_img *
255).astype(np.uint8))
        normalized_img.save(output_path)

# Loop through each image in the resized dataset
for filename in os.listdir(resized_dataset_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        # Create the full path for the resized and normalized images
        resized_image_path = os.path.join(resized_dataset_path,
filename)
        normalized_image_path = os.path.join(normalized_dataset_path,
filename)

        # Normalize and save the image
        normalize_and_save(resized_image_path, normalized_image_path)

# Display some normalized images for reference
num_images_to_display = 4
selected_normalized_images = os.listdir(normalized_dataset_path)
[:num_images_to_display]

# Create a subplot with rows and columns
rows = 1
columns = num_images_to_display
fig, axs = plt.subplots(rows, columns, figsize=(12, 3))

# Display each normalized image in the subplot
for i in range(columns):
    img_path = os.path.join(normalized_dataset_path,
selected_normalized_images[i])
    img = mpimg.imread(img_path)
    axs[i].imshow(img, cmap='gray')
    axs[i].axis('off')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```



SHARPENING

```
from PIL import Image, ImageFilter

# Path to the normalized dataset
normalized_dataset_path =
"/content/drive/MyDrive/Normalized_HEART_US_IMAGE_FETUS"

# Path to the location to store sharpened images
sharpened_dataset_path =
"/content/drive/MyDrive/Sharpened_HEART_US_IMAGE_FETUS"

# Create the new directory if it doesn't exist
if not os.path.exists(sharpened_dataset_path):
    os.makedirs(sharpened_dataset_path)

# Function to sharpen and save images
def sharpen_and_save(image_path, output_path):
    with Image.open(image_path) as img:
        # Apply a sharpening filter
        sharpened_img = img.filter(ImageFilter.SHARPEN)

        # Save the sharpened image
        sharpened_img.save(output_path)

# Loop through each image in the normalized dataset
for filename in os.listdir(normalized_dataset_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        # Create the full path for the normalized and sharpened images
        normalized_image_path = os.path.join(normalized_dataset_path,
filename)
        sharpened_image_path = os.path.join(sharpened_dataset_path,
filename)

        # Sharpen and save the image
        sharpen_and_save(normalized_image_path, sharpened_image_path)

# Display some sharpened images for reference
num_images_to_display = 4
selected_sharpened_images = os.listdir(sharpened_dataset_path)
[:num_images_to_display]
```

```python
# Create a subplot with rows and columns
rows = 1
columns = num_images_to_display
fig, axs = plt.subplots(rows, columns, figsize=(12, 3))

# Display each sharpened image in the subplot
for i in range(columns):
    img_path = os.path.join(sharpened_dataset_path,
selected_sharpened_images[i])
    img = mpimg.imread(img_path)
    axs[i].imshow(img, cmap='gray')
    axs[i].axis('off')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```

CONTRAST ADJUSTMENT

```python
from PIL import Image, ImageFilter, ImageEnhance

# Path to the sharpened dataset
sharpened_dataset_path =
"/content/drive/MyDrive/Sharpened_HEART_US_IMAGE_FETUS"

# Path to the location to store contrast-stretched images
contrast_stretched_dataset_path =
"/content/drive/MyDrive/Contrast_Stretched_HEART_US_IMAGE_FETUS"

# Create the new directory if it doesn't exist
if not os.path.exists(contrast_stretched_dataset_path):
    os.makedirs(contrast_stretched_dataset_path)

# Function to apply contrast stretching and save images
def contrast_stretch_and_save(image_path, output_path,
enhancement_factor=1.5):
    with Image.open(image_path) as img:
```

```
        # Apply contrast stretching using ImageEnhance
        contrast = ImageEnhance.Contrast(img)
        stretched_img = contrast.enhance(enhancement_factor)

        # Save the contrast-stretched image
        stretched_img.save(output_path)

# Loop through each image in the sharpened dataset
for filename in os.listdir(sharpened_dataset_path):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        # Create the full path for the sharpened and contrast-
stretched images
        sharpened_image_path = os.path.join(sharpened_dataset_path,
filename)
        contrast_stretched_image_path =
os.path.join(contrast_stretched_dataset_path, filename)

        # Contrast stretch and save the image
        contrast_stretch_and_save(sharpened_image_path,
contrast_stretched_image_path)

# Display some contrast-stretched images for reference
num_images_to_display = 4
selected_contrast_stretched_images =
os.listdir(contrast_stretched_dataset_path)[:num_images_to_display]

# Create a subplot with rows and columns
rows = 1
columns = num_images_to_display
fig, axs = plt.subplots(rows, columns, figsize=(12, 3))

# Display each contrast-stretched image in the subplot
for i in range(columns):
    img_path = os.path.join(contrast_stretched_dataset_path,
selected_contrast_stretched_images[i])
    img = mpimg.imread(img_path)
    axs[i].imshow(img, cmap='gray')
    axs[i].axis('off')

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```

CLUSTER ANALYSIS

K-Means Clustering Before PCA

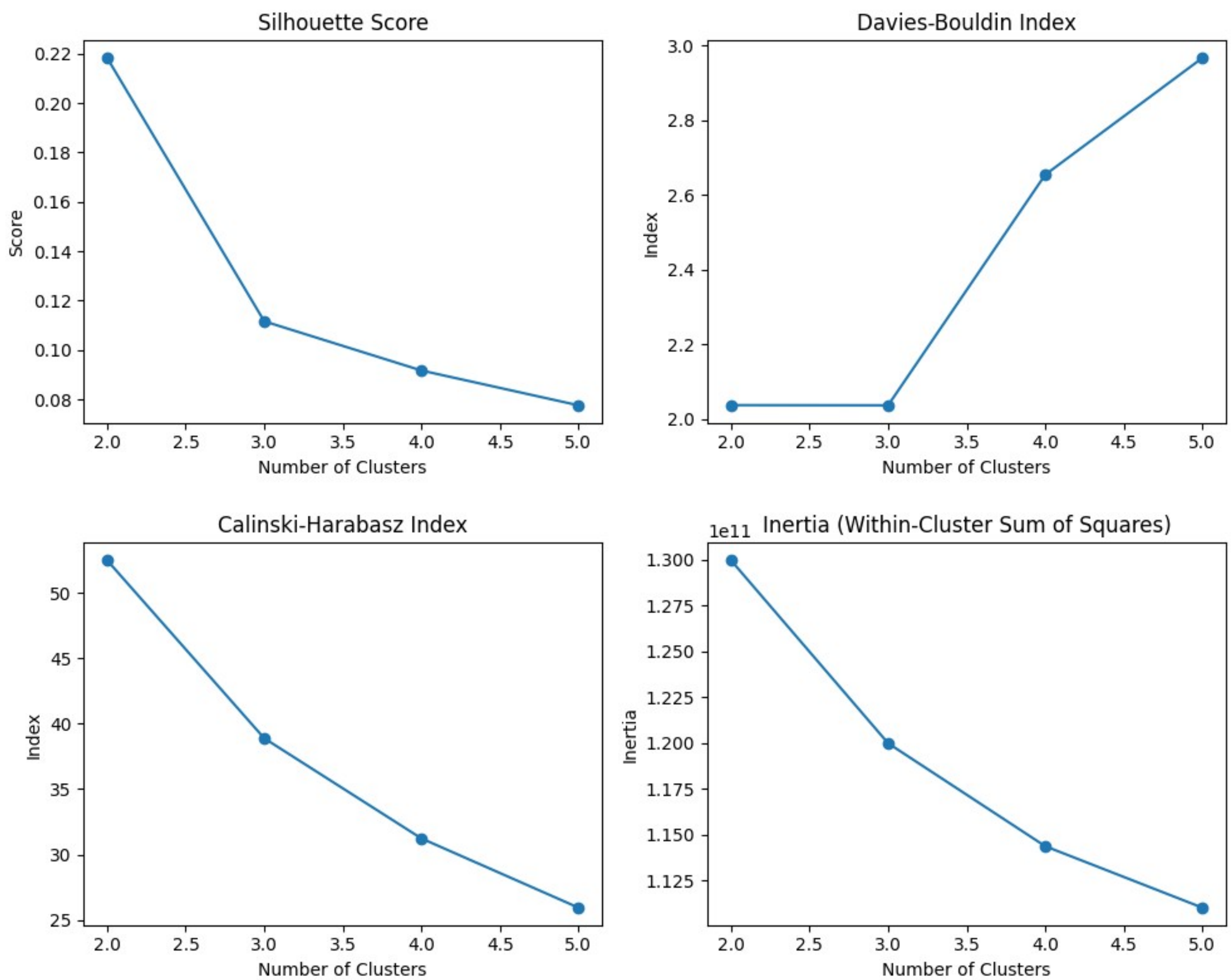
```
import os
import cv2
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score
import matplotlib.pyplot as plt

# Define the path to your image dataset
dataset_path ="/content/drive/MyDrive/HEART US IMAGE FETUS"

# Function to load and preprocess images
def load_and_preprocess_images(dataset_path, image_shape):
    image_list = []
    for filename in os.listdir(dataset_path):
        if filename.endswith('.jpg'):
            img = cv2.imread(os.path.join(dataset_path, filename))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to
RGB format
            img = cv2.resize(img, image_shape)  # Resize the image
            image_list.append(img)
    return image_list

# Load and preprocess the images
image_shape = (256, 256)
images = load_and_preprocess_images(dataset_path, image_shape)

# Convert the list of images to a numpy array
image_data = np.array(images).reshape(len(images), -1)

# Range of clusters to try
cluster_range = range(2, 6)
```

```
# Lists to store metrics for plotting
silhouette_scores = []
davies_bouldin_indices = []
calinski_harabasz_indices = []
inertia_values = []

# Perform K-Means Clustering for different cluster numbers
for n_clusters in cluster_range:
    # Perform K-Means Clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    cluster_labels = kmeans.fit_predict(image_data)

    # Calculate validation measures
    silhouette_avg = silhouette_score(image_data, cluster_labels)
    davies_bouldin_index = davies_bouldin_score(image_data,
cluster_labels)
    calinski_harabasz_index = calinski_harabasz_score(image_data,
cluster_labels)
    inertia = kmeans.inertia_

    # Append metrics to lists
    silhouette_scores.append(silhouette_avg)
    davies_bouldin_indices.append(davies_bouldin_index)
    calinski_harabasz_indices.append(calinski_harabasz_index)
    inertia_values.append(inertia)

    # Print the results for each iteration
    print(f"\nClusters: {n_clusters}")
    print(f"Silhouette Score: {silhouette_avg}")
    print(f"Davies-Bouldin Index: {davies_bouldin_index}")
    print(f"Calinski-Harabasz Index: {calinski_harabasz_index}")
    print(f"Inertia (Within-Cluster Sum of Squares): {inertia}")


    # Plotting the results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Silhouette Score
axes[0, 0].plot(cluster_range, silhouette_scores, marker='o')
axes[0, 0].set_title('Silhouette Score')
axes[0, 0].set_xlabel('Number of Clusters')
axes[0, 0].set_ylabel('Score')

# Davies-Bouldin Index
axes[0, 1].plot(cluster_range, davies_bouldin_indices, marker='o')
axes[0, 1].set_title('Davies-Bouldin Index')
axes[0, 1].set_xlabel('Number of Clusters')
axes[0, 1].set_ylabel('Index')

# Calinski-Harabasz Index
```

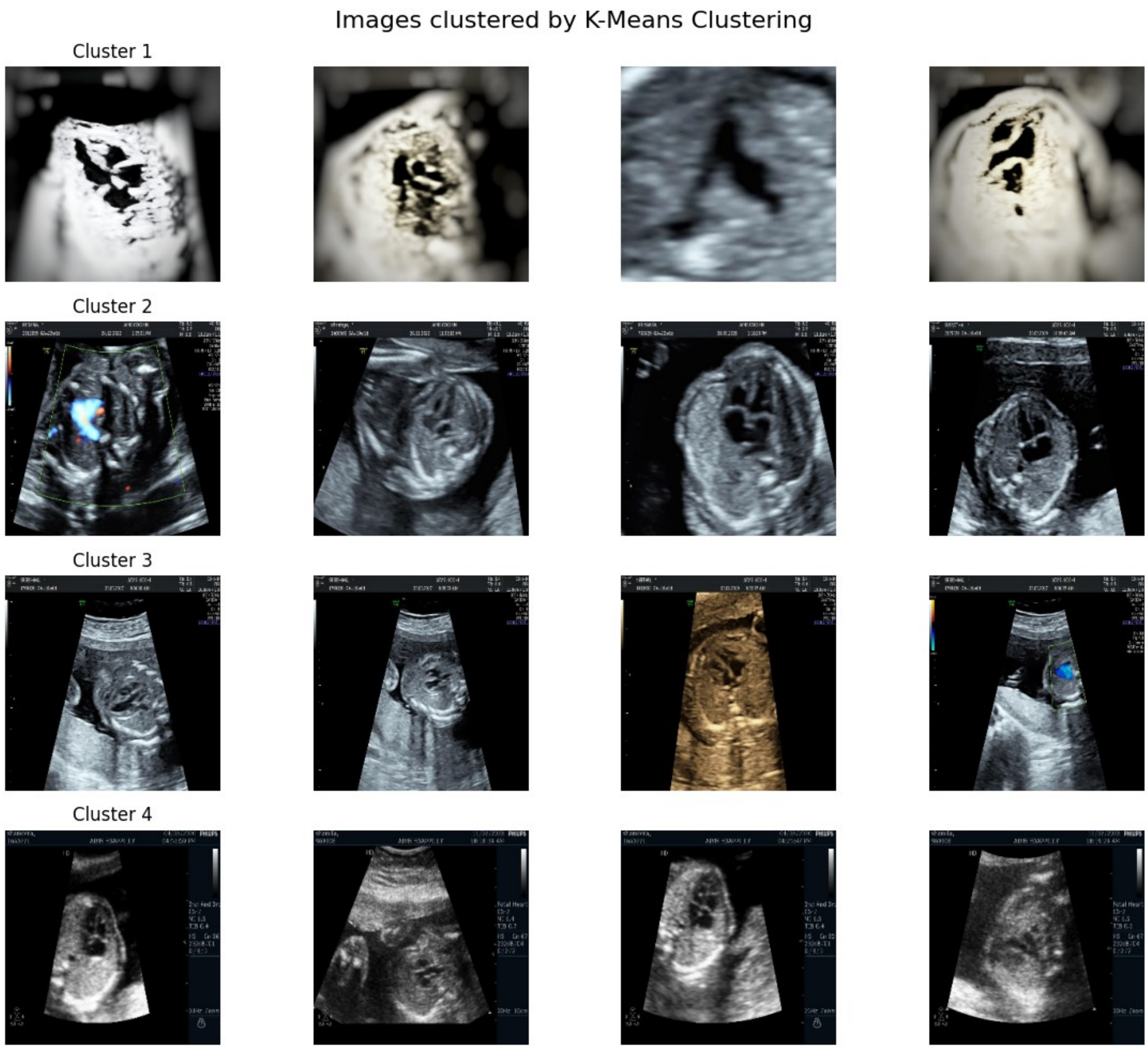
```
axes[1, 0].plot(cluster_range, calinski_harabasz_indices, marker='o')
axes[1, 0].set_title('Calinski-Harabasz Index')
axes[1, 0].set_xlabel('Number of Clusters')
axes[1, 0].set_ylabel('Index')

# Inertia
axes[1, 1].plot(cluster_range, inertia_values, marker='o')
axes[1, 1].set_title('Inertia (Within-Cluster Sum of Squares)')
axes[1, 1].set_xlabel('Number of Clusters')
axes[1, 1].set_ylabel('Inertia')

plt.tight_layout()
plt.show()


# Function to plot images with K-Means cluster labels
def plot_kmeans_clusters(images, cluster_labels, n_clusters):
    plt.figure(figsize=(12, 10))
    for i in range(n_clusters):
        cluster_images = np.array(images)[cluster_labels == i]
        for j, img in enumerate(cluster_images[:4]):  # Plot the first
4 images in each cluster
            plt.subplot(n_clusters, 4, i * 4 + j + 1)
            plt.imshow(img)
            plt.axis('off')
            if j == 0:
                plt.title(f"Cluster {i+1}")
    plt.suptitle("Images clustered by K-Means Clustering",
fontsize=16)
    plt.tight_layout()
    plt.show()

# Perform K-Means Clustering with the optimal number of clusters
(choose the one with the best metrics)
optimal_n_clusters = 4 # Change this based on the optimal number of
clusters from  metrics evaluation

# Perform K-Means Clustering
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=0)
kmeans_labels = kmeans.fit_predict(image_data)

# Plot images with K-Means cluster labels
plot_kmeans_clusters(images, kmeans_labels, optimal_n_clusters)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

```
Clusters: 2
Silhouette Score: 0.21831584385645292
Davies-Bouldin Index: 2.0366010319213683
Calinski-Harabasz Index: 52.47932861076172
Inertia (Within-Cluster Sum of Squares): 129982164882.10896

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(


Clusters: 3
Silhouette Score: 0.1116258412019826
Davies-Bouldin Index: 2.036148998803733
Calinski-Harabasz Index: 38.88229007875694
Inertia (Within-Cluster Sum of Squares): 119995016069.84192

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(


Clusters: 4
Silhouette Score: 0.09169623173001427
Davies-Bouldin Index: 2.6543221381193884
Calinski-Harabasz Index: 31.23418685763801
Inertia (Within-Cluster Sum of Squares): 114372860125.29172

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(


Clusters: 5
Silhouette Score: 0.0775871354154438
Davies-Bouldin Index: 2.9673434854456824
Calinski-Harabasz Index: 25.942897772548857
Inertia (Within-Cluster Sum of Squares): 111016820321.53664
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



Hierarchical Clustering Before PCA

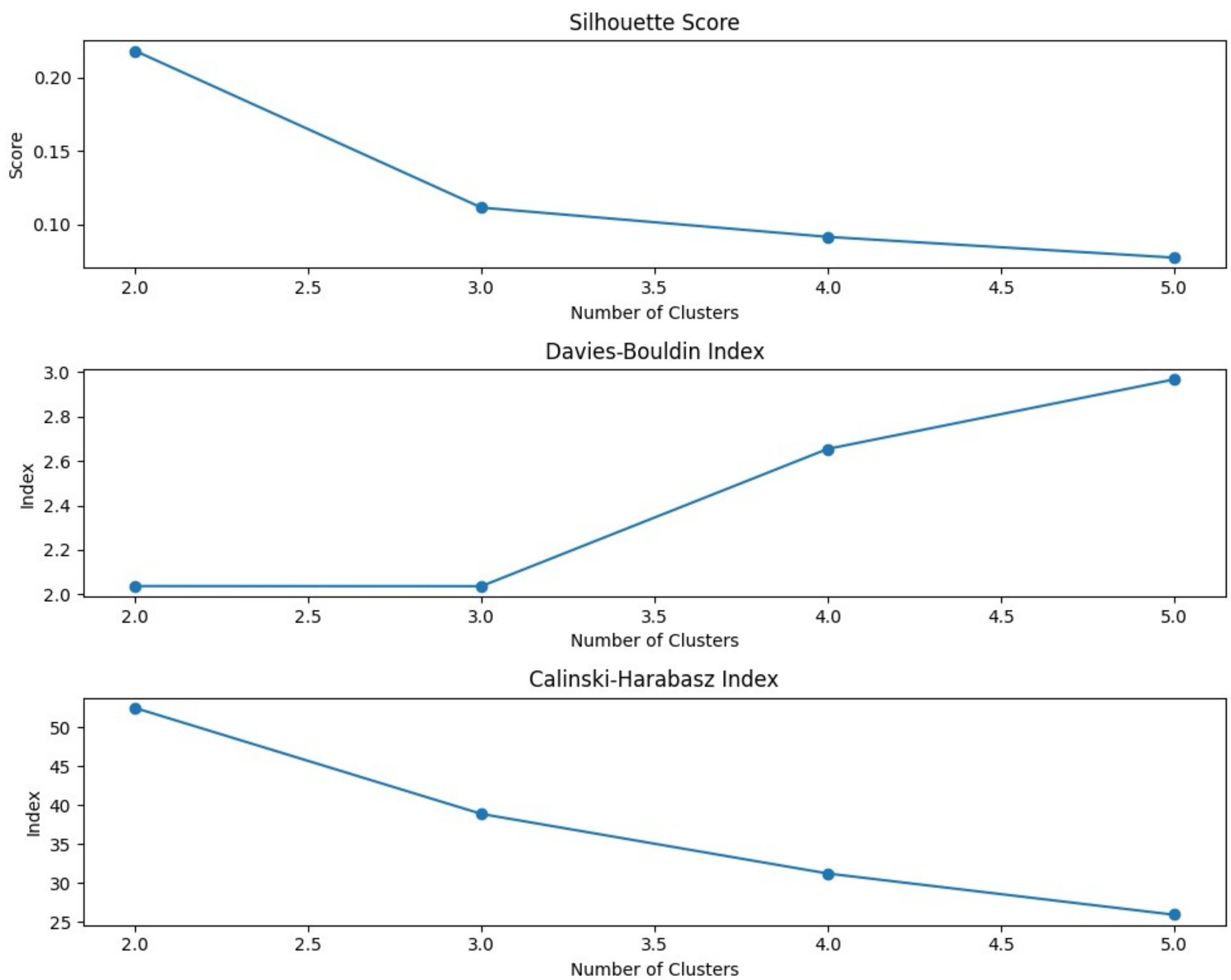
```
import os
import cv2
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score
from scipy.cluster.hierarchy import linkage
import matplotlib.pyplot as plt

# Define the path to your image dataset
dataset_path ="/content/drive/MyDrive/HEART US IMAGE FETUS"
```

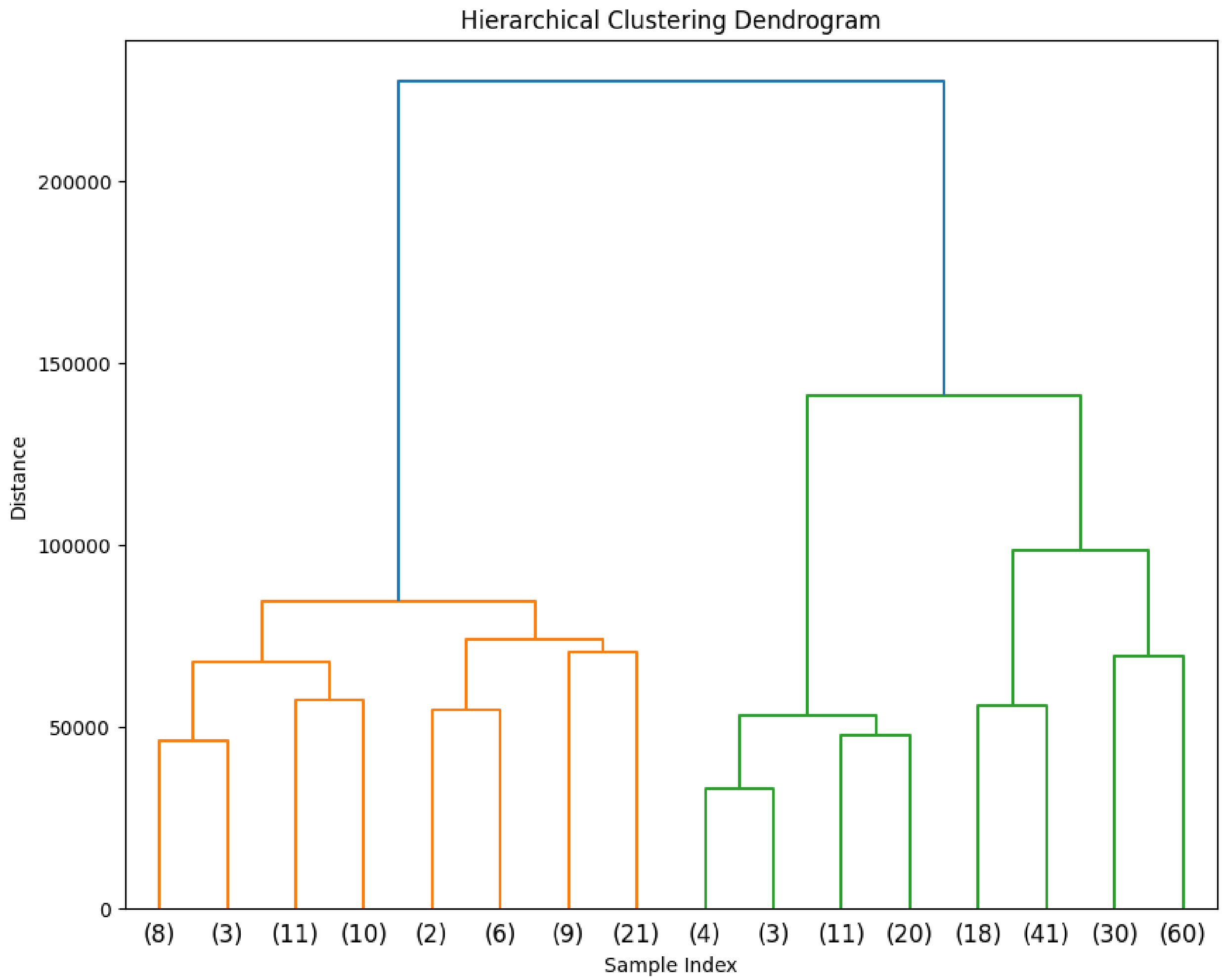
```
# Function to load and preprocess images
def load_and_preprocess_images(dataset_path, image_shape):
    image_list = []
    for filename in os.listdir(dataset_path):
        if filename.endswith('.jpg'):
            img = cv2.imread(os.path.join(dataset_path, filename))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to
RGB format
            img = cv2.resize(img, image_shape)  # Resize the image
            image_list.append(img)
    return image_list

# Load and preprocess the images
image_shape = (256, 256)
images = load_and_preprocess_images(dataset_path, image_shape)

# Convert the list of images to a numpy array
image_data = np.array(images).reshape(len(images), -1)

# Number of clusters to print
n_clusters_to_print = 4

# Perform Hierarchical Clustering for different cluster numbers
for n_clusters in cluster_range:
    # Perform Hierarchical Clustering
    linkage_matrix = linkage(image_data, method='ward')
    cluster_labels = AgglomerativeClustering(n_clusters=n_clusters,
linkage='ward').fit_predict(image_data)

    # Calculate validation measures
    silhouette_avg = silhouette_score(image_data, cluster_labels)
    davies_bouldin_index = davies_bouldin_score(image_data,
cluster_labels)
    calinski_harabasz_index = calinski_harabasz_score(image_data,
cluster_labels)

    # Print the results for each iteration
    print(f"\nClusters: {n_clusters}")
    print(f"Silhouette Score: {silhouette_avg}")
    print(f"Davies-Bouldin Index: {davies_bouldin_index}")
    print(f"Calinski-Harabasz Index: {calinski_harabasz_index}")


    import matplotlib.pyplot as plt

# Plotting the results
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 8))

# Silhouette Score
axes[0].plot(cluster_range, silhouette_scores, marker='o')
```

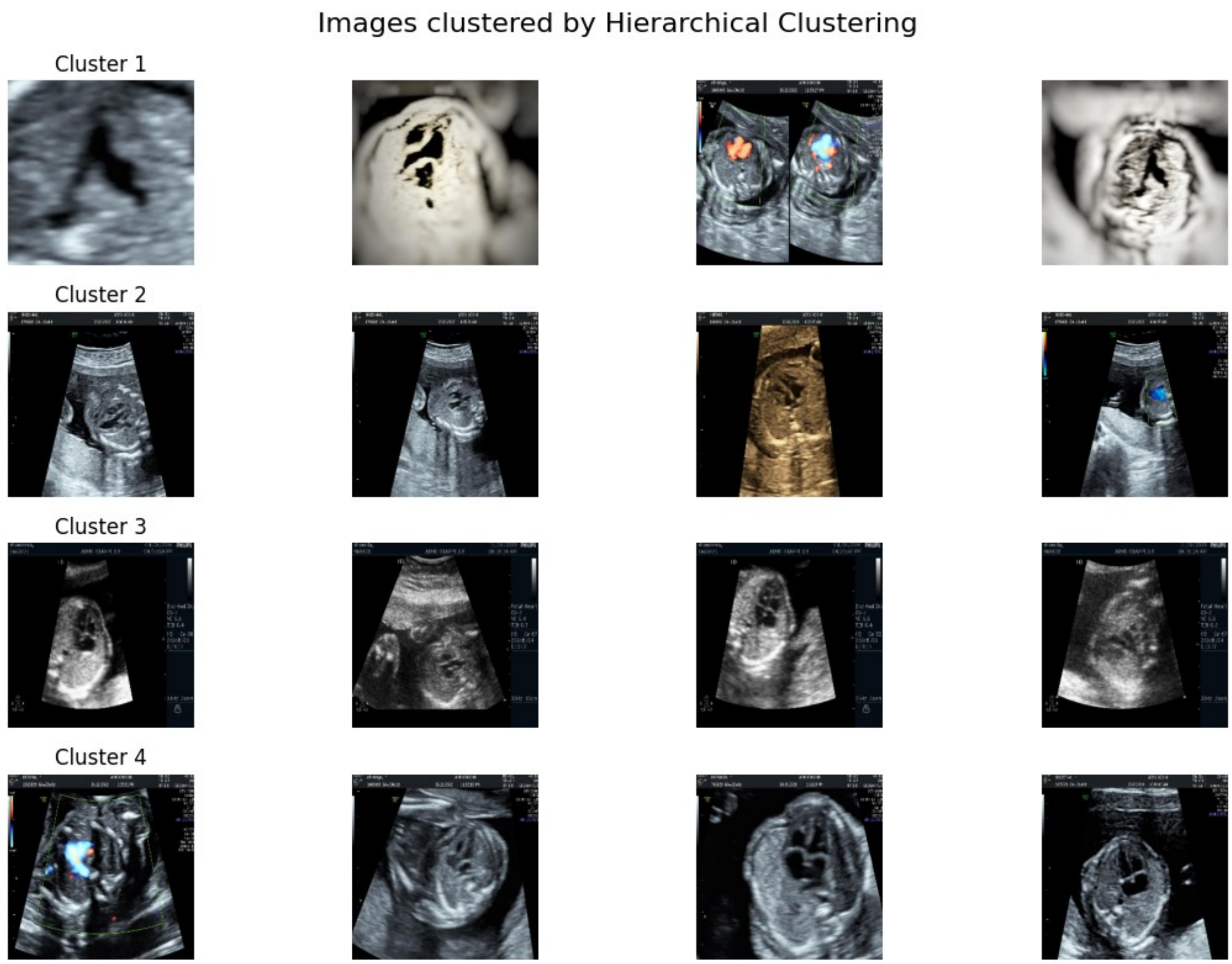
```
axes[0].set_title('Silhouette Score')
axes[0].set_xlabel('Number of Clusters')
axes[0].set_ylabel('Score')

# Davies-Bouldin Index
axes[1].plot(cluster_range, davies_bouldin_indices, marker='o')
axes[1].set_title('Davies-Bouldin Index')
axes[1].set_xlabel('Number of Clusters')
axes[1].set_ylabel('Index')

# Calinski-Harabasz Index
axes[2].plot(cluster_range, calinski_harabasz_indices, marker='o')
axes[2].set_title('Calinski-Harabasz Index')
axes[2].set_xlabel('Number of Clusters')
axes[2].set_ylabel('Index')

plt.tight_layout()
plt.show()



import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram

# Visualize the dendrogram
plt.figure(figsize=(10, 8))
dendrogram(linkage_matrix, truncate_mode='level', p=3)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Plot images with cluster labels for 5 clusters
def plot_clusters(images, cluster_labels, n_clusters):
    plt.figure(figsize=(12, 8))
    for i in range(n_clusters):
        cluster_images = np.array(images)[cluster_labels == i]
        for j, img in enumerate(cluster_images[:4]):  # Plot the first
4 images in each cluster
            plt.subplot(n_clusters, 4, i * 4 + j + 1)
            plt.imshow(img)
            plt.axis('off')
            if j == 0:
                plt.title(f"Cluster {i+1}")
    plt.suptitle("Images clustered by Hierarchical Clustering",
fontsize=16)
    plt.tight_layout()
    plt.show()
```

```python
# Plot images with cluster labels for 5 clusters
plot_clusters(images, cluster_labels, n_clusters_to_print)
```

```
Clusters: 2
Silhouette Score: 0.2118013531330989
Davies-Bouldin Index: 2.086092478685209
Calinski-Harabasz Index: 50.44523228771401

Clusters: 3
Silhouette Score: 0.10989746772157044
Davies-Bouldin Index: 2.0620912233509574
Calinski-Harabasz Index: 37.65589167609462

Clusters: 4
Silhouette Score: 0.08222164933862167
Davies-Bouldin Index: 2.8479662492596383
Calinski-Harabasz Index: 29.557593549365208

Clusters: 5
Silhouette Score: 0.08437409196898955
Davies-Bouldin Index: 3.145034498265756
Calinski-Harabasz Index: 24.77350995415099
```


Silhouette Score
Score
0.20
0.15
0.10
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Clusters
Davies-Bouldin Index
Index
3.0
2.8
2.6
2.4
2.2
2.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Clusters
Calinski-Harabasz Index
Index
50
45
40
35
30
25
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Clusters

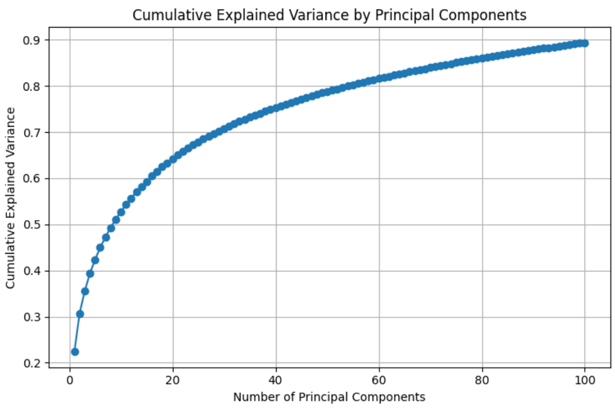

Hierarchical Clustering Dendrogram
Distance
200000
150000
100000
50000
0
(8)
(3)
(11)
(10)
(2)
(6)
(9)
(21)
(4)
(3)
(11)
(20)
(18)
(41)
(30)
(60)
Sample Index

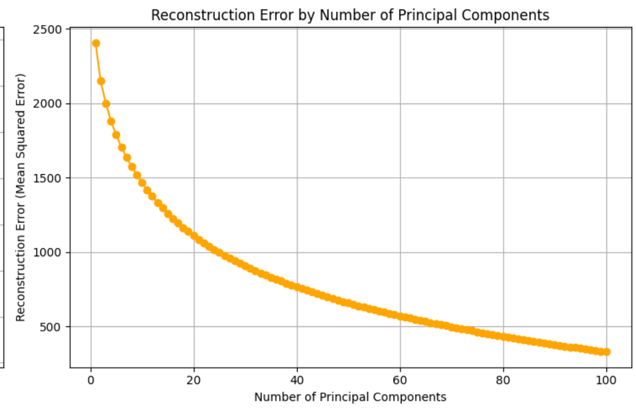


Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error

# Load the preprocessed images
image_shape = (256, 256)
images = load_and_preprocess_images(dataset_path, image_shape)

# Convert the list of images to a numpy array
image_data = np.array(images).reshape(len(images), -1)

# Define the range of principal components to try
max_num_components = min(image_data.shape[0], image_data.shape[1],
100)
num_components_range = range(1, max_num_components + 1)

# Initialize lists to store results
cumulative_explained_variance = []
```

```
reconstruction_errors = []

# Iterate over different numbers of principal components
for num_components in num_components_range:
    # Initialize PCA
    pca = PCA(n_components=num_components)

    # Fit PCA on the image data
    pca.fit(image_data)

    # Transform the image data using the fitted PCA
    image_data_pca = pca.transform(image_data)

    # Reconstruct the original data from the reduced representation
    reconstructed_data = pca.inverse_transform(image_data_pca)

    # Calculate cumulative explained variance

cumulative_explained_variance.append(np.sum(pca.explained_variance_rat
io_))

    # Calculate reconstruction error (mean squared error)
    reconstruction_error = mean_squared_error(image_data,
reconstructed_data)
    reconstruction_errors.append(reconstruction_error)

# Plot the explained variance ratio and reconstruction error
plt.figure(figsize=(15, 5))

# Explained Variance Ratio
plt.subplot(1, 2, 1)
plt.plot(num_components_range, cumulative_explained_variance,
marker='o')
plt.title('Cumulative Explained Variance by Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)

# Reconstruction Error
plt.subplot(1, 2, 2)
plt.plot(num_components_range, reconstruction_errors, marker='o',
color='orange')
plt.title('Reconstruction Error by Number of Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Reconstruction Error (Mean Squared Error)')
plt.grid(True)

plt.tight_layout()
plt.show()
```





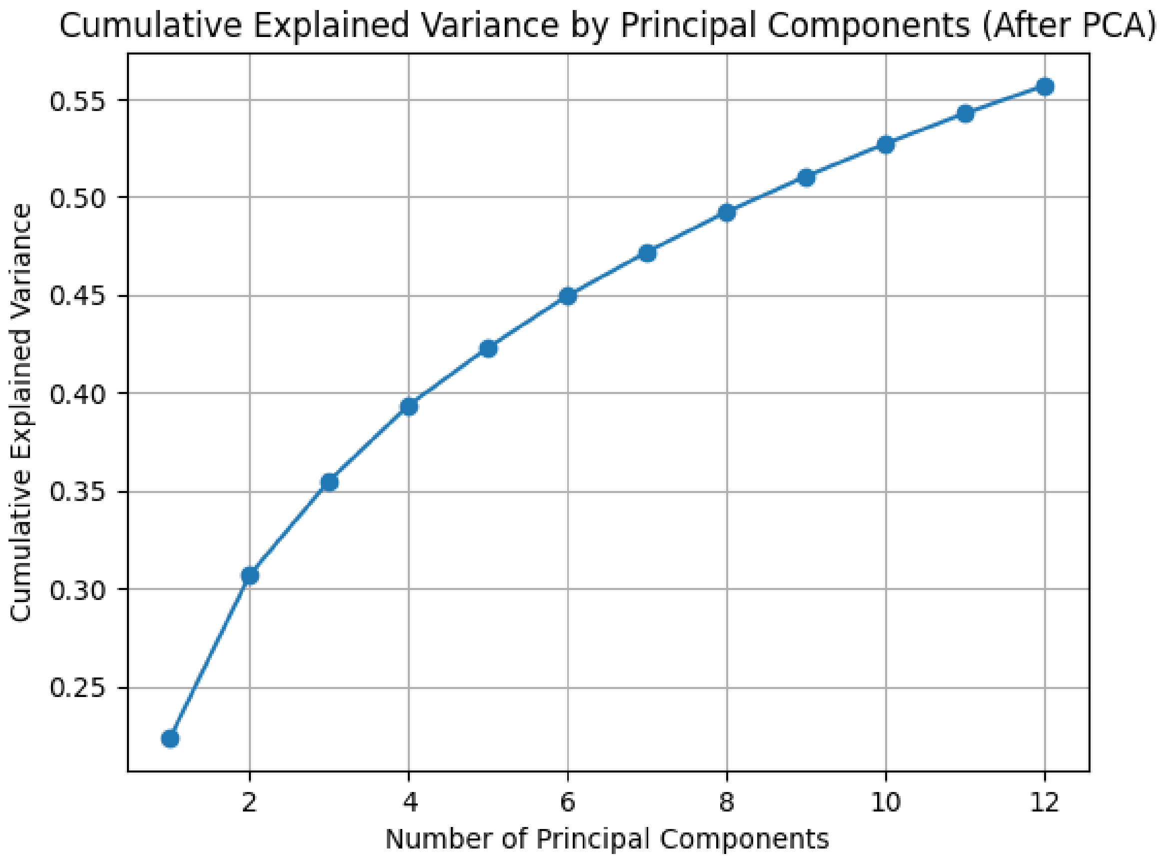
K-Means Clustering After PCA

```
# Principal Component Analysis (PCA)
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

# Load the preprocessed and normalized images
image_shape = (256, 256)
images = load_and_preprocess_images(dataset_path, image_shape)

# Convert the list of images to a numpy array
image_data = np.array(images).reshape(len(images), -1)

# Reduce the number of components
num_components = min(image_data.shape[0], image_data.shape[1], 12)

# Initialize PCA with the chosen number of components
pca = PCA(n_components=num_components)

# Fit PCA on the normalized image data
image_data_pca = pca.fit_transform(image_data)

# Reconstruct the original data from the reduced representation
reconstructed_data = pca.inverse_transform(image_data_pca)

# Display the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Plot the explained variance ratio
plt.plot(range(1, num_components + 1), cumulative_explained_variance,
marker='o')
plt.title('Cumulative Explained Variance by Principal Components
```

```
(After PCA)')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()

# Print the explained variance ratio for each component
for i, ratio in enumerate(explained_variance_ratio, 1):
    print(f"Explained Variance Ratio for Component {i}: {ratio:.4f}")

# Print the cumulative explained variance for each component
for i, cumulative_ratio in enumerate(cumulative_explained_variance,
1):
    print(f"Cumulative Explained Variance for Components 1 to {i}:
{cumulative_ratio:.4f}")

# Print the reconstruction error (mean squared error) after PCA
reconstruction_error_after_pca = mean_squared_error(image_data,
reconstructed_data)
print(f"\nReconstruction Error (Mean Squared Error) After PCA:
{reconstruction_error_after_pca:.4f}")


# K-Means Clustering After PCA
# Use the reduced representation obtained from PCA for K-Means
clustering

# Range of clusters to try
cluster_range_after_pca = range(2, 6)

# Lists to store metrics for plotting
silhouette_scores_after_pca = []
davies_bouldin_indices_after_pca = []
calinski_harabasz_indices_after_pca = []
inertia_values_after_pca = []

# Perform K-Means Clustering for different cluster numbers after PCA
for n_clusters_after_pca in cluster_range_after_pca:
    # Perform K-Means Clustering
    kmeans_after_pca = KMeans(n_clusters=n_clusters_after_pca,
random_state=0)
    cluster_labels_after_pca =
kmeans_after_pca.fit_predict(image_data_pca)

    # Calculate validation measures
    silhouette_avg_after_pca = silhouette_score(image_data_pca,
cluster_labels_after_pca)
    davies_bouldin_index_after_pca =
davies_bouldin_score(image_data_pca, cluster_labels_after_pca)
    calinski_harabasz_index_after_pca =
```

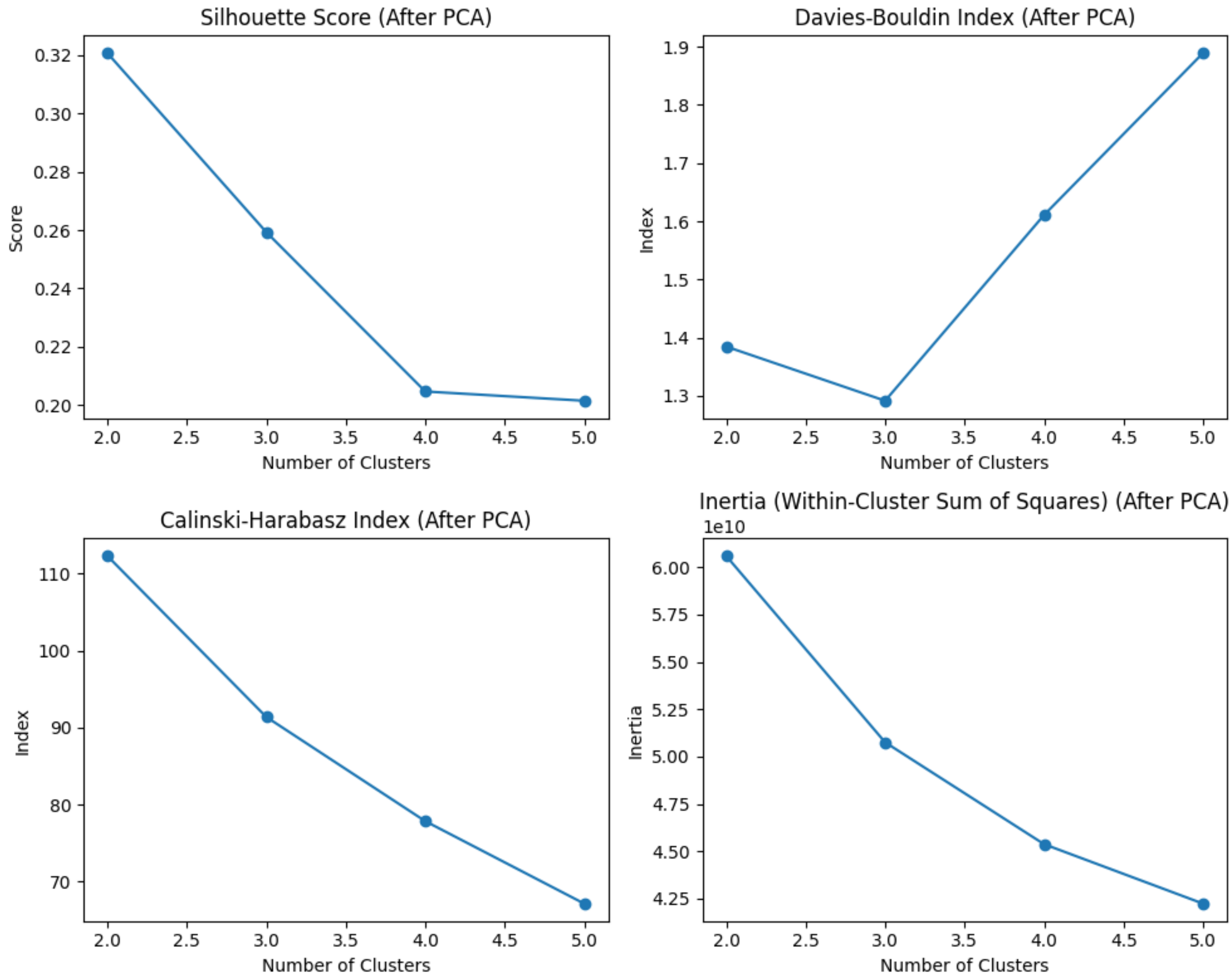
```
calinski_harabasz_score(image_data_pca, cluster_labels_after_pca)
    inertia_after_pca = kmeans_after_pca.inertia_

    # Append metrics to lists
    silhouette_scores_after_pca.append(silhouette_avg_after_pca)

davies_bouldin_indices_after_pca.append(davies_bouldin_index_after_pca
)

calinski_harabasz_indices_after_pca.append(calinski_harabasz_index_aft
er_pca)
    inertia_values_after_pca.append(inertia_after_pca)

    # Print the results for each iteration after PCA
    print(f"\nClusters (after PCA): {n_clusters_after_pca}")
    print(f"Silhouette Score: {silhouette_avg_after_pca}")
    print(f"Davies-Bouldin Index: {davies_bouldin_index_after_pca}")
    print(f"Calinski-Harabasz Index:
{calinski_harabasz_index_after_pca}")
    print(f"Inertia (Within-Cluster Sum of Squares):
{inertia_after_pca}")

# Plotting the results after PCA
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Silhouette Score
axes[0, 0].plot(cluster_range_after_pca, silhouette_scores_after_pca,
marker='o')
axes[0, 0].set_title('Silhouette Score (After PCA)')
axes[0, 0].set_xlabel('Number of Clusters')
axes[0, 0].set_ylabel('Score')

# Davies-Bouldin Index
axes[0, 1].plot(cluster_range_after_pca,
davies_bouldin_indices_after_pca, marker='o')
axes[0, 1].set_title('Davies-Bouldin Index (After PCA)')
axes[0, 1].set_xlabel('Number of Clusters')
axes[0, 1].set_ylabel('Index')

# Calinski-Harabasz Index
axes[1, 0].plot(cluster_range_after_pca,
calinski_harabasz_indices_after_pca, marker='o')
axes[1, 0].set_title('Calinski-Harabasz Index (After PCA)')
axes[1, 0].set_xlabel('Number of Clusters')
axes[1, 0].set_ylabel('Index')

# Inertia
axes[1, 1].plot(cluster_range_after_pca, inertia_values_after_pca,
marker='o')
axes[1, 1].set_title('Inertia (Within-Cluster Sum of Squares) (After
```

```
PCA)')
axes[1, 1].set_xlabel('Number of Clusters')
axes[1, 1].set_ylabel('Inertia')

plt.tight_layout()
plt.show()

# Function to plot images with K-Means cluster labels after PCA
def plot_kmeans_clusters_after_pca(images, cluster_labels_after_pca,
n_clusters_after_pca):
    plt.figure(figsize=(12, 10))
    for i in range(n_clusters_after_pca):
        cluster_images = np.array(images)[cluster_labels_after_pca ==
i]
        for j, img in enumerate(cluster_images[:4]):  # Plot the first
4 images in each cluster
            plt.subplot(n_clusters_after_pca, 4, i * 4 + j + 1)
            plt.imshow(img)
            plt.axis('off')
            if j == 0:
                plt.title(f"Cluster {i+1}")
    plt.suptitle("Images clustered by K-Means Clustering After PCA",
fontsize=16)
    plt.tight_layout()
    plt.show()

# Perform K-Means Clustering with the optimal number of clusters after
PCA
optimal_n_clusters_after_pca = 4  # Change this based on the optimal
number of clusters from metrics evaluation

# Perform K-Means Clustering after PCA
kmeans_after_pca = KMeans(n_clusters=optimal_n_clusters_after_pca,
random_state=0)
kmeans_labels_after_pca = kmeans_after_pca.fit_predict(image_data_pca)

# Plot images with K-Means cluster labels after PCA
plot_kmeans_clusters_after_pca(images, kmeans_labels_after_pca,
optimal_n_clusters_after_pca)
```




```
Explained Variance Ratio for Component 1: 0.2239
Explained Variance Ratio for Component 2: 0.0831
Explained Variance Ratio for Component 3: 0.0481
Explained Variance Ratio for Component 4: 0.0385
Explained Variance Ratio for Component 5: 0.0295
Explained Variance Ratio for Component 6: 0.0265
Explained Variance Ratio for Component 7: 0.0223
Explained Variance Ratio for Component 8: 0.0205
Explained Variance Ratio for Component 9: 0.0181
Explained Variance Ratio for Component 10: 0.0167
Explained Variance Ratio for Component 11: 0.0155
Explained Variance Ratio for Component 12: 0.0142
Cumulative Explained Variance for Components 1 to 1: 0.2239
Cumulative Explained Variance for Components 1 to 2: 0.3070
Cumulative Explained Variance for Components 1 to 3: 0.3551
Cumulative Explained Variance for Components 1 to 4: 0.3936
Cumulative Explained Variance for Components 1 to 5: 0.4231
Cumulative Explained Variance for Components 1 to 6: 0.4496
Cumulative Explained Variance for Components 1 to 7: 0.4719
Cumulative Explained Variance for Components 1 to 8: 0.4923
Cumulative Explained Variance for Components 1 to 9: 0.5105
Cumulative Explained Variance for Components 1 to 10: 0.5272
Cumulative Explained Variance for Components 1 to 11: 0.5427
```

```
Cumulative Explained Variance for Components 1 to 12: 0.5568

Reconstruction Error (Mean Squared Error) After PCA: 1374.6607

Clusters (after PCA): 2
Silhouette Score: 0.32078068706981205
Davies-Bouldin Index: 1.3847942966732052
Calinski-Harabasz Index: 112.27024746848204
Inertia (Within-Cluster Sum of Squares): 60594972681.136826

Clusters (after PCA): 3
Silhouette Score: 0.25930023543800274
Davies-Bouldin Index: 1.2917181180199082
Calinski-Harabasz Index: 91.37644133156405
Inertia (Within-Cluster Sum of Squares): 50755147907.43767

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(


Clusters (after PCA): 4
Silhouette Score: 0.20461041660912876
Davies-Bouldin Index: 1.6108482715663852
Calinski-Harabasz Index: 77.86092021372241
Inertia (Within-Cluster Sum of Squares): 45378064105.871185

Clusters (after PCA): 5
Silhouette Score: 0.20142929809609408
Davies-Bouldin Index: 1.8890287298819533
Calinski-Harabasz Index: 67.16392883688395
Inertia (Within-Cluster Sum of Squares): 42240792914.61592

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

Hierarchical Clustering After PCA

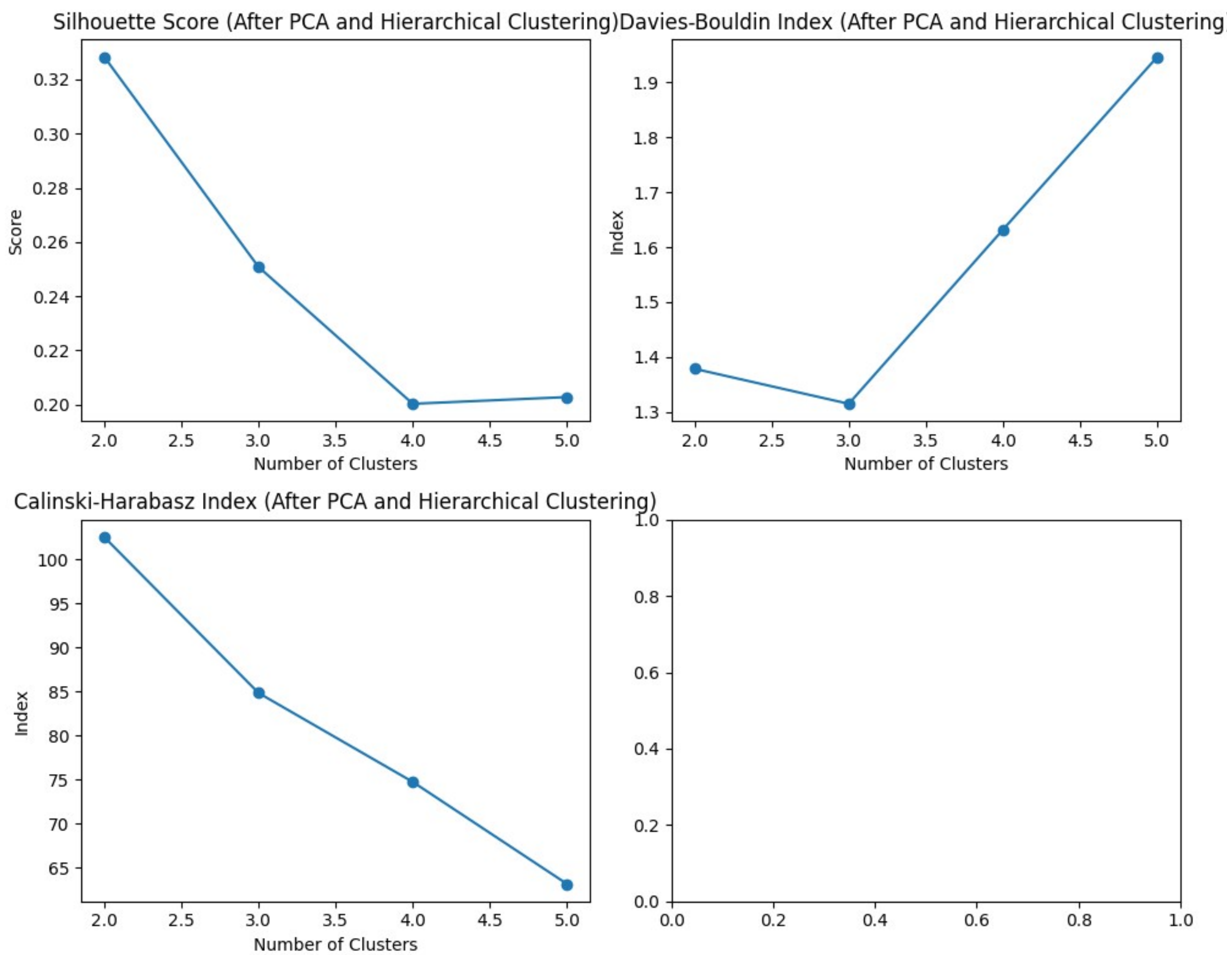
```
from sklearn.cluster import AgglomerativeClustering

# **Hierarchical Clustering After PCA**
# Range of clusters to try
cluster_range_after_pca_hierarchical = range(2, 6)

# Lists to store metrics for plotting
silhouette_scores_after_pca_hierarchical = []
davies_bouldin_indices_after_pca_hierarchical = []
calinski_harabasz_indices_after_pca_hierarchical = []

# Perform Hierarchical Clustering for different cluster numbers after
```

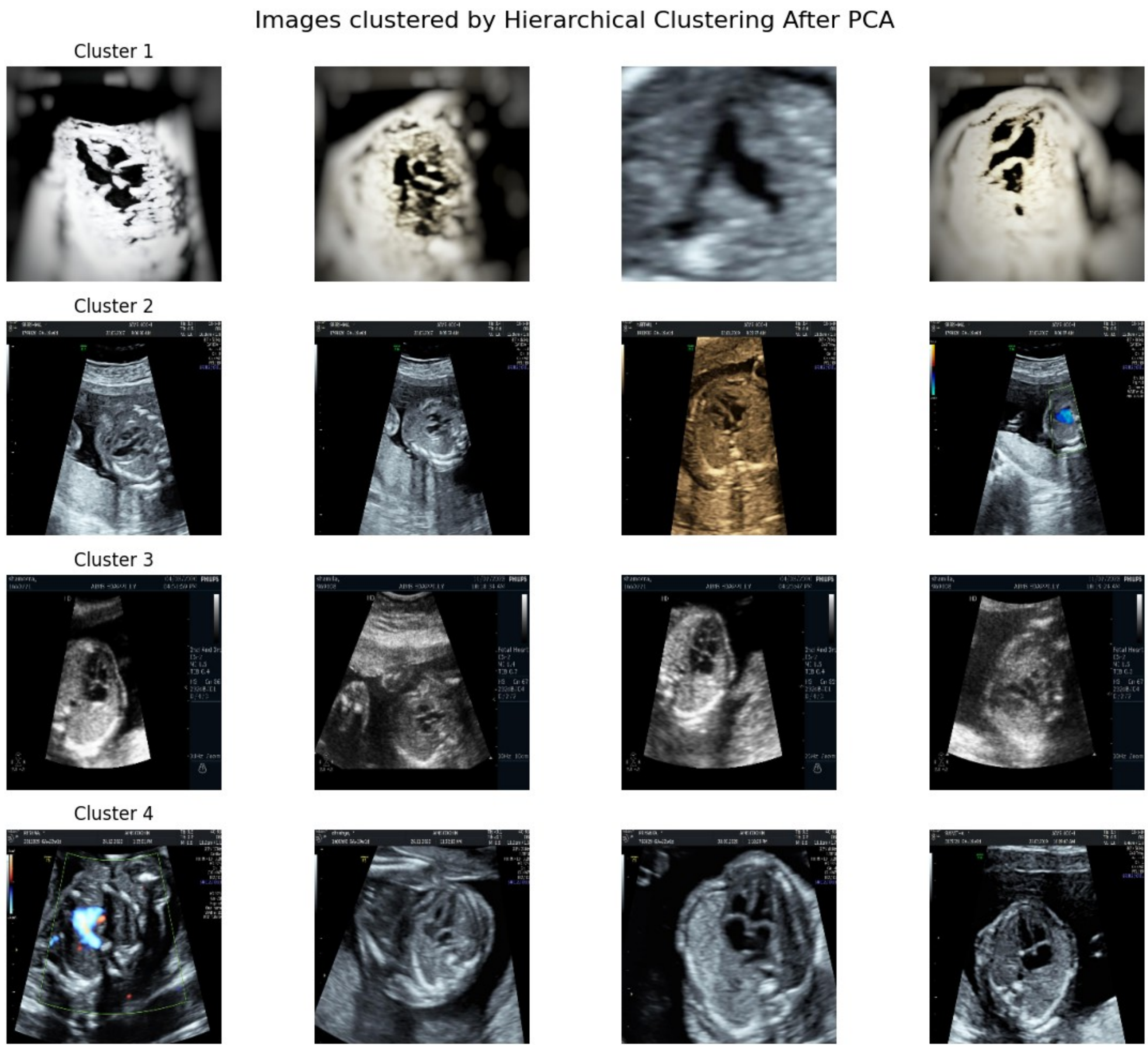
```
PCA
for n_clusters_after_pca_hierarchical in
cluster_range_after_pca_hierarchical:
    # Perform Hierarchical Clustering
    hierarchical_after_pca =
AgglomerativeClustering(n_clusters=n_clusters_after_pca_hierarchical)
    cluster_labels_after_pca_hierarchical =
hierarchical_after_pca.fit_predict(image_data_pca)

    # Calculate validation measures
    silhouette_avg_after_pca_hierarchical =
silhouette_score(image_data_pca,
cluster_labels_after_pca_hierarchical)
    davies_bouldin_index_after_pca_hierarchical =
davies_bouldin_score(image_data_pca,
cluster_labels_after_pca_hierarchical)
    calinski_harabasz_index_after_pca_hierarchical =
calinski_harabasz_score(image_data_pca,
cluster_labels_after_pca_hierarchical)

    # Append metrics to lists

silhouette_scores_after_pca_hierarchical.append(silhouette_avg_after_p
ca_hierarchical)

davies_bouldin_indices_after_pca_hierarchical.append(davies_bouldin_in
dex_after_pca_hierarchical)

calinski_harabasz_indices_after_pca_hierarchical.append(calinski_harab
asz_index_after_pca_hierarchical)

    # Print the results for each iteration after PCA and Hierarchical
Clustering
    print(f"\nClusters (after PCA and Hierarchical Clustering):
{n_clusters_after_pca_hierarchical}")
    print(f"Silhouette Score:
{silhouette_avg_after_pca_hierarchical}")
    print(f"Davies-Bouldin Index:
{davies_bouldin_index_after_pca_hierarchical}")
    print(f"Calinski-Harabasz Index:
{calinski_harabasz_index_after_pca_hierarchical}")

# Plotting the results after PCA and Hierarchical Clustering
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Silhouette Score
axes[0, 0].plot(cluster_range_after_pca_hierarchical,
silhouette_scores_after_pca_hierarchical, marker='o')
axes[0, 0].set_title('Silhouette Score (After PCA and Hierarchical
Clustering)')
```

```
axes[0, 0].set_xlabel('Number of Clusters')
axes[0, 0].set_ylabel('Score')

# Davies-Bouldin Index
axes[0, 1].plot(cluster_range_after_pca_hierarchical,
davies_bouldin_indices_after_pca_hierarchical, marker='o')
axes[0, 1].set_title('Davies-Bouldin Index (After PCA and Hierarchical
Clustering)')
axes[0, 1].set_xlabel('Number of Clusters')
axes[0, 1].set_ylabel('Index')

# Calinski-Harabasz Index
axes[1, 0].plot(cluster_range_after_pca_hierarchical,
calinski_harabasz_indices_after_pca_hierarchical, marker='o')
axes[1, 0].set_title('Calinski-Harabasz Index (After PCA and
Hierarchical Clustering)')
axes[1, 0].set_xlabel('Number of Clusters')
axes[1, 0].set_ylabel('Index')

plt.tight_layout()
plt.show()

# Function to plot images with Hierarchical cluster labels after PCA
def plot_hierarchical_clusters_after_pca(images,
cluster_labels_after_pca_hierarchical,
n_clusters_after_pca_hierarchical):
    plt.figure(figsize=(12, 10))
    for i in range(n_clusters_after_pca_hierarchical):
        cluster_images_hierarchical = np.array(images)
[cluster_labels_after_pca_hierarchical == i]
        for j, img_hierarchical in
enumerate(cluster_images_hierarchical[:4]):  # Plot the first 4 images
in each cluster
            plt.subplot(n_clusters_after_pca_hierarchical, 4, i * 4 +
j + 1)
            plt.imshow(img_hierarchical)
            plt.axis('off')
            if j == 0:
                plt.title(f"Cluster {i+1}")
    plt.suptitle("Images clustered by Hierarchical Clustering After
PCA", fontsize=16)
    plt.tight_layout()
    plt.show()

# Perform Hierarchical Clustering with the optimal number of clusters
after PCA
optimal_n_clusters_after_pca_hierarchical = 4  # Change this based on
the optimal number of clusters from metrics evaluation

# Perform Hierarchical Clustering after PCA
```

```
hierarchical_after_pca =
AgglomerativeClustering(n_clusters=optimal_n_clusters_after_pca_hierar
chical)
hierarchical_labels_after_pca =
hierarchical_after_pca.fit_predict(image_data_pca)

# Plot images with Hierarchical cluster labels after PCA
plot_hierarchical_clusters_after_pca(images,
hierarchical_labels_after_pca,
optimal_n_clusters_after_pca_hierarchical)
```

```
Clusters (after PCA and Hierarchical Clustering): 2
Silhouette Score: 0.3283172198992113
Davies-Bouldin Index: 1.378474766144968
Calinski-Harabasz Index: 102.51767646892412

Clusters (after PCA and Hierarchical Clustering): 3
Silhouette Score: 0.25081513987976933
Davies-Bouldin Index: 1.31471585865028
Calinski-Harabasz Index: 84.84436603168552

Clusters (after PCA and Hierarchical Clustering): 4
Silhouette Score: 0.20032026804782224
Davies-Bouldin Index: 1.6320601027968022
Calinski-Harabasz Index: 74.75762137889707

Clusters (after PCA and Hierarchical Clustering): 5
Silhouette Score: 0.2027966344066904
Davies-Bouldin Index: 1.9467104388303718
Calinski-Harabasz Index: 63.186862359089794
```


Silhouette Score (After PCA and Hierarchical Clustering)
Score
Number of Clusters
Davies-Bouldin Index (After PCA and Hierarchical Clustering)
Index
Number of Clusters
Calinski-Harabasz Index (After PCA and Hierarchical Clustering)
Index
Number of Clusters



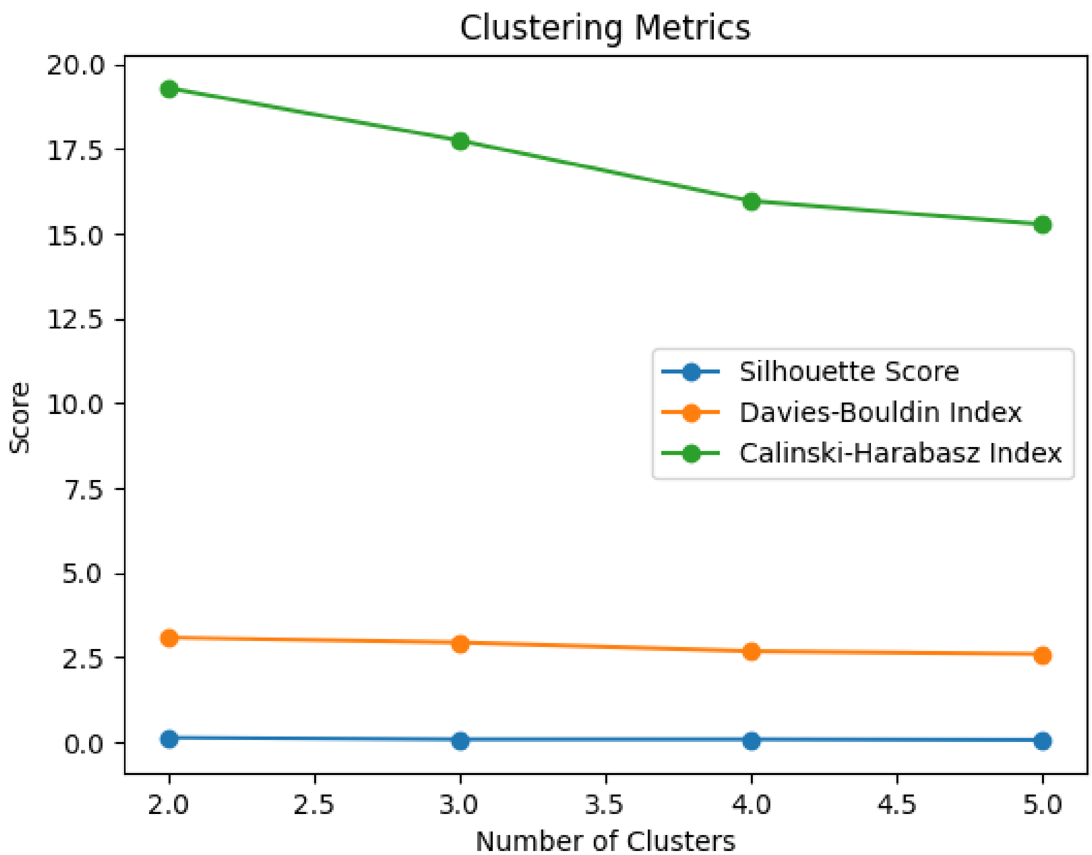

```
import os
import cv2
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score
import matplotlib.pyplot as plt

# Define paths
original_dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"
mask_dataset_path =
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segment
ationClass"

# Function to load and preprocess segmented mask images
```

```
def load_and_preprocess_masks(dataset_path, image_shape):
    mask_list = []
    for filename in os.listdir(dataset_path):
        if filename.endswith('.png'):
            img = cv2.imread(os.path.join(dataset_path, filename),
cv2.IMREAD_GRAYSCALE)  # Load as grayscale
            img = cv2.resize(img, image_shape)  # Resize the mask
image
            mask_list.append(img)
    return mask_list

# Load and preprocess the segmented mask images
mask_shape = (256, 256)
masks = load_and_preprocess_masks(mask_dataset_path, mask_shape)

# Convert the list of masks to a numpy array
mask_data = np.array(masks).reshape(len(masks), -1)

# Define clustering function using the mask data
def perform_clustering(data, cluster_range):
    # Define lists to store metrics
    silhouette_scores = []
    davies_bouldin_indices = []
    calinski_harabasz_indices = []

    # Perform clustering for different cluster numbers
    for n_clusters in cluster_range:
        # Perform K-Means Clustering
        kmeans = KMeans(n_clusters=n_clusters, random_state=0)
        cluster_labels = kmeans.fit_predict(data)

        # Calculate validation measures
        silhouette_avg = silhouette_score(data, cluster_labels)
        davies_bouldin_index = davies_bouldin_score(data,
cluster_labels)
        calinski_harabasz_index = calinski_harabasz_score(data,
cluster_labels)

        # Append metrics to lists
        silhouette_scores.append(silhouette_avg)
        davies_bouldin_indices.append(davies_bouldin_index)
        calinski_harabasz_indices.append(calinski_harabasz_index)

        # Print the results for each iteration
        print(f"\nClusters: {n_clusters}")
        print(f"Silhouette Score: {silhouette_avg}")
        print(f"Davies-Bouldin Index: {davies_bouldin_index}")
        print(f"Calinski-Harabasz Index: {calinski_harabasz_index}")

    # Plot the results
```

```python
    plt.plot(cluster_range, silhouette_scores, marker='o', 
label='Silhouette Score')
    plt.plot(cluster_range, davies_bouldin_indices, marker='o', 
label='Davies-Bouldin Index')
    plt.plot(cluster_range, calinski_harabasz_indices, marker='o', 
label='Calinski-Harabasz Index')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Score')
    plt.title('Clustering Metrics')
    plt.legend()
    plt.show()

# Perform clustering on the mask data
cluster_range = range(2, 6)  # Adjust as needed
perform_clustering(mask_data, cluster_range)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will 
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly 
to suppress the warning
  warnings.warn(


Clusters: 2
Silhouette Score: 0.135270996628909
Davies-Bouldin Index: 3.088019793559278
Calinski-Harabasz Index: 19.305082526154386

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will 
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly 
to suppress the warning
  warnings.warn(


Clusters: 3
Silhouette Score: 0.0832238563473491
Davies-Bouldin Index: 2.9391619168362855
Calinski-Harabasz Index: 17.756692206257163

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will 
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly 
to suppress the warning
  warnings.warn(


Clusters: 4
Silhouette Score: 0.08254175747304328
Davies-Bouldin Index: 2.6893354911207403
Calinski-Harabasz Index: 15.970156819785982
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(


Clusters: 5
Silhouette Score: 0.064298214299714
Davies-Bouldin Index: 2.60396321416787
Calinski-Harabasz Index: 15.282118715639866
```



PIX2PIX GAN SEGMENTATION

```
import numpy as np # linear algebra
import tensorflow as tf # for tensorflow based registration
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import skimage
import os
from cv2 import imread, createCLAHE # read and equalize images
import cv2
from glob import glob
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import pathlib
import time
import datetime
from IPython import display
from skimage.util import montage as montage2d


mask_dataset_path =
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS"

# Print all mask paths
print("Mask paths:")
for filename in os.listdir(mask_dataset_path):
    print(os.path.join(mask_dataset_path, filename))
```

```
Mask paths:
/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/labelmap
.txt
/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segmenta
tionObject
/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segmenta
tionClass
/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/ImageSet
s
```

```
import os

# Define the path to the dataset
mask_dataset_path =
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS"

# Print all mask file paths
print("Mask file paths:")
for filename in os.listdir(mask_dataset_path):
    file_path = os.path.join(mask_dataset_path, filename)
    if os.path.isfile(file_path):
        print(file_path)
```

```
Mask file paths:
/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/labelmap
.txt
```

```
import os
import cv2
from glob import glob

# Define the paths
original_dataset_path =
"/content/drive/MyDrive/Sharpened_HEART_US_IMAGE_FETUS"
mask_dataset_path =
```

```
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segment
ationClass"

# Function to load and resize images
def load_and_resize_images(image_paths, mask_paths, image_shape):
    images = []
    masks = []
    for image_path, mask_path in zip(image_paths, mask_paths):
        # Load and resize the image
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB
format
        img = cv2.resize(img, image_shape)
        images.append(img)

        # Load and resize the mask
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)  # Read
mask as grayscale
        mask = cv2.resize(mask, image_shape)
        masks.append(mask)
    return images, masks

# Get all image paths
image_paths = glob(os.path.join(original_dataset_path, '*.jpg'))

# Get corresponding mask paths
mask_paths = []
for image_path in image_paths:
    image_filename = os.path.basename(image_path)
    mask_path = os.path.join(mask_dataset_path,
image_filename.replace('.jpg', '.png'))
    mask_paths.append(mask_path)

# Define the desired image shape
#image_shape = (256, 256)  # Set your desired image shape

# Load and resize images with masks
images, masks = load_and_resize_images(image_paths, mask_paths,
image_shape)

# Check the number of loaded images
print("Number of images loaded:", len(images))
print("Number of masks loaded:", len(masks))
```

```
Number of images loaded: 257
Number of masks loaded: 257
```

```
import os
from glob import glob
import numpy as np
```

```python
from skimage.io import imread as imread_raw
from skimage.transform import resize
from tqdm import tqdm
import warnings

warnings.filterwarnings('ignore', category=UserWarning,
module='skimage')  # Suppress skimage warnings



OUT_DIM = (256, 256)

def imread(in_path, apply_clahe=False):
    img_data = imread_raw(in_path)
    n_img = (255 * resize(img_data, OUT_DIM, mode='constant')).clip(0,
255).astype(np.float64)
    if apply_clahe:
        clahe_tool = createCLAHE(clipLimit=2.0, tileGridSize=(16, 16))
        n_img = clahe_tool.apply(n_img)
    return np.expand_dims(n_img, -1)

# Loading the ultrasound images and resizing them
ultrasound_data = []
for u_path, m_path in tqdm(ultrasound_images):
    u_img = imread(u_path)
    m_img = imread(m_path)
    ultrasound_data.append((u_img, m_img))
```

```
100%|██████████| 257/257 [00:24<00:00, 10.49it/s]
```

```python
img_vol, seg_vol = [], []

for img_path, m_path in tqdm(ultrasound_images):
    img_vol.append(np.squeeze(imread(img_path)))
    seg_vol.append(np.squeeze(imread(m_path, apply_clahe=False)))

img_vol = np.stack(img_vol, 0)
seg_vol = np.stack(seg_vol, 0)

print('Images:', img_vol.shape, 'Masks:', seg_vol.shape)
```

```
100%|██████████| 257/257 [00:24<00:00, 10.43it/s]
```

```
Images: (257, 256, 256, 3) Masks: (257, 256, 256, 3)
```

```python
import os
from glob import glob
from tqdm import tqdm
import numpy as np
from PIL import Image
```

```
# Define dataset paths
original_dataset_path =
"/content/drive/MyDrive/Sharpened_HEART_US_IMAGE_FETUS"
mask_dataset_path =
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segment
ationClass"

# Define the target size for resizing
target_size = (256,256)  # Specify the desired width and height

# Get paths of ultrasound images (JPG format)
ultrasound_image_paths = glob(os.path.join(original_dataset_path,
'*.jpg'))
print('ULTRASOUND_IMAGES:', len(ultrasound_image_paths))

# Get paths of corresponding mask images (PNG format)
mask_image_paths = [os.path.join(mask_dataset_path,
os.path.splitext(os.path.basename(img_path))[0] + '.png') for img_path
in ultrasound_image_paths]

# Initialize lists to store resized image volumes and segmentation
volumes
img_vol = []
seg_vol = []

# Iterate through image paths and corresponding mask paths
for img_path, mask_path in tqdm(zip(ultrasound_image_paths,
mask_image_paths), total=len(ultrasound_image_paths)):
    # Load image and resize
    img = Image.open(img_path).resize(target_size, Image.ANTIALIAS)

    # Load mask and resize
    mask = Image.open(mask_path).resize(target_size, Image.ANTIALIAS)

    # Convert to numpy arrays
    img = np.array(img)
    mask = np.array(mask)

    # Append resized image and mask to lists
    img_vol.append(img)
    seg_vol.append(mask)

# Convert lists to numpy arrays
img_vol = np.stack(img_vol, axis=0)
seg_vol = np.stack(seg_vol, axis=0)

print('Images:', img_vol.shape, 'Segmentations:', seg_vol.shape)
```

```
ULTRASOUND_IMAGES: 257
```

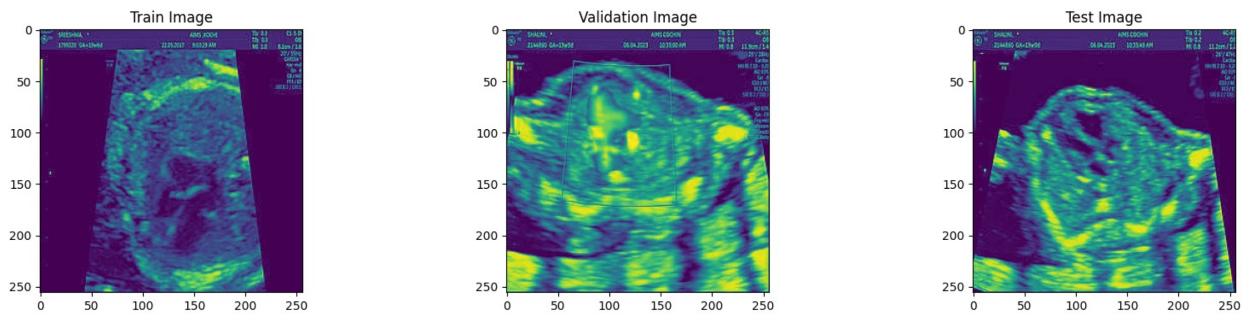
```
  0%|          | 0/257 [00:00<?, ?it/s]<ipython-input-25-
0a20db12e2e8>:28: DeprecationWarning: ANTIALIAS is deprecated and will
be removed in Pillow 10 (2023-07-01). Use LANCZOS or
Resampling.LANCZOS instead.
  img = Image.open(img_path).resize(target_size, Image.ANTIALIAS)
<ipython-input-25-0a20db12e2e8>:31: DeprecationWarning: ANTIALIAS is
deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS
or Resampling.LANCZOS instead.
  mask = Image.open(mask_path).resize(target_size, Image.ANTIALIAS)
100%|██████████| 257/257 [00:07<00:00, 32.38it/s]

Images: (257, 256, 256, 3) Segmentations: (257, 256, 256, 3)
```

```
import numpy as np
import matplotlib.pyplot as plt

# Select the first image and its corresponding mask
t_img, m_img = img_vol[0], seg_vol[0]

# Plotting the image and its mask
fig, (ax_img, ax_mask) = plt.subplots(1, 2, figsize=(12, 6))

# Display the ultrasound image
ax_img.imshow(t_img, cmap='gray')  # Assuming ultrasound images are
grayscale
ax_img.set_title('Ultrasound Image')

# Display the mask
ax_mask.imshow(m_img, cmap='gray')
ax_mask.set_title('Mask Image')

plt.show()
```


```
from sklearn.model_selection import train_test_split

# Assuming img_vol and seg_vol are your image and segmentation volumes
# Adjust the test_size parameter for your desired split
train_vol, test_vol, train_seg, test_seg = train_test_split((img_vol -
127.0) / 127.0,
                                                            (seg_vol >
127).astype(np.float32),

test_size=0.2,  # 20% for test set

random_state=2018)

# Split the remaining data (80% of the original) into train and
validation sets
train_vol, val_vol, train_seg, val_seg = train_test_split(train_vol,
train_seg,

test_size=0.25,  # 25% of the 80% for validation set

random_state=2018)

print('Train:', train_vol.shape, 'Validation:', val_vol.shape,
'Test:', test_vol.shape)
print('Train Set Mean:', train_vol.mean(), 'Train Set Max:',
train_vol.max())
print('Validation Set Max:', val_vol.max())
print('Test Set Mean:', test_vol.mean(), 'Test Set Max:',
test_vol.max())

# Plotting code for visualizing samples from each set
```

```python
fig, axes = plt.subplots(1, 3, figsize=(20, 4))

axes[0].imshow(train_vol[0, :, :, 0])
axes[0].set_title('Train Image')
axes[1].imshow(val_vol[0, :, :, 0])
axes[1].set_title('Validation Image')
axes[2].imshow(test_vol[0, :, :, 0])
axes[2].set_title('Test Image')
plt.show()
```

```
Train: (153, 256, 256, 3) Validation: (52, 256, 256, 3) Test: (52,
256, 256, 3)
Train Set Mean: -0.5410883464416989 Train Set Max: 1.0078740157480315
Validation Set Max: 1.0078740157480315
Test Set Mean: -0.5288979114245187 Test Set Max: 1.0078740157480315
```




```python
from keras.preprocessing.image import ImageDataGenerator

# Define the data augmentation parameters
dg_args = dict(
    featurewise_center=False,
    samplewise_center=False,
    rotation_range=5,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.01,
    zoom_range=[0.8, 1.2],  # Random zoom
    horizontal_flip=True,
    vertical_flip=False,
    fill_mode='nearest',
    data_format='channels_last'
)

# Additional data augmentation techniques such as elastic deformations
and contrast adjustments
additional_augmentation = dict(
    elastic_deformation=True,
    contrast_adjustment=True
)
```

```
# Initialize the ImageDataGenerator object
image_gen = ImageDataGenerator(**dg_args)

# Define a generator function to yield augmented data with additional
transformations
def gen_augmented_pairs(in_vol, in_seg, batch_size=1, train=True):
    while True:
        if train:
            seed = np.random.choice(range(9999))
            g_vol = image_gen.flow(in_vol, batch_size=batch_size,
seed=seed)
            g_seg = image_gen.flow(in_seg, batch_size=batch_size,
seed=seed)
            for i_vol, i_seg in zip(g_vol, g_seg):
                # Apply additional transformations
                if additional_augmentation['elastic_deformation']:
                    # Apply elastic deformations
                    i_vol, i_seg = apply_elastic_deformations(i_vol,
i_seg)
                if additional_augmentation['contrast_adjustment']:
                    # Apply contrast adjustments
                    i_vol, i_seg = apply_contrast_adjustment(i_vol,
i_seg)
                yield i_vol, i_seg
        else:
            seed = 0
            g_vol = image_gen.flow(in_vol, batch_size=batch_size,
seed=seed)
            g_seg = image_gen.flow(in_seg, batch_size=batch_size,
seed=seed)
            for i_vol, i_seg in zip(g_vol, g_seg):
                yield i_vol, i_seg

# Generate augmented data with additional transformations for training
train_generator = gen_augmented_pairs(train_vol, train_seg,
batch_size=1, train=True)

# Generate augmented data with additional transformations for
validation
val_generator = gen_augmented_pairs(val_vol, val_seg, batch_size=1,
train=True)

# Generate augmented data for testing without additional
transformations
test_generator = gen_augmented_pairs(test_vol, test_seg, batch_size=1,
train=False)

# Define the generators for training, validation, and testing
train_generator = gen_augmented_pairs(train_vol, train_seg,
```

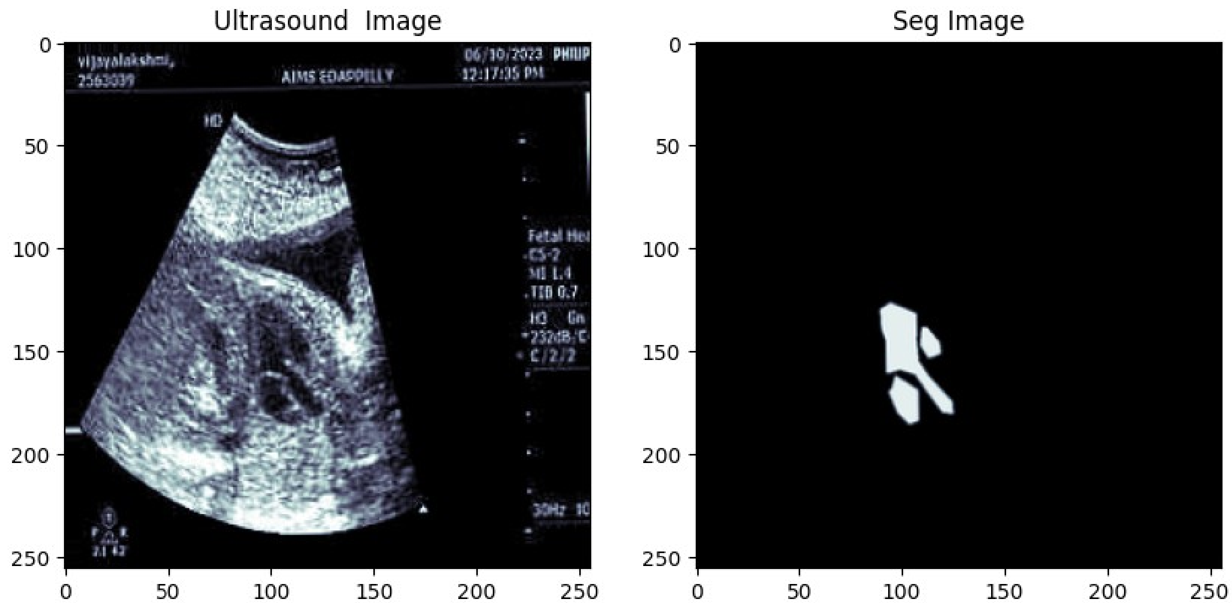
```
batch_size=1, train=True)
val_generator = gen_augmented_pairs(val_vol, val_seg, batch_size=1,
train=True)
test_generator = gen_augmented_pairs(test_vol, test_seg, batch_size=1,
train=False)

# Generate a batch of augmented data for training
train_X, train_Y = next(train_generator)
# Generate a batch of augmented data for testing
test_X, test_Y = next(test_generator)

# Print the shapes of the generated data
print("Training Data Shape:", train_X.shape, train_Y.shape)
print("Testing Data Shape:", test_X.shape, test_Y.shape)
```

```
Training Data Shape: (1, 256, 256, 3) (1, 256, 256, 3)
Testing Data Shape: (1, 256, 256, 3) (1, 256, 256, 3)
```

```
import matplotlib.pyplot as plt
import numpy as np

# Define a function to visualize a montage of 2D images
def montage2d(images):
    # Calculate the number of rows and columns for the montage
    num_images = images.shape[0]
    num_cols = int(np.ceil(np.sqrt(num_images)))
    num_rows = int(np.ceil(num_images / num_cols))

    # Create an empty array for the montage
    montage = np.zeros((num_rows * images.shape[1], num_cols *
images.shape[2]))

    # Fill in the montage with images
    for i in range(num_images):
        row = i // num_cols
        col = i % num_cols
        montage[row * images.shape[1] : (row + 1) * images.shape[1],
col * images.shape[2] : (col + 1) * images.shape[2]] = images[i, :, :]

    return montage

# Generate a batch of augmented data for training
train_X, train_Y = next(train_generator)

# Generate a batch of augmented data for testing
test_X, test_Y = next(test_generator)


fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 5))
ax1.imshow(montage2d(train_X[:, :, :, 0]), cmap = 'bone')
ax1.set_title('Ultrasound  Image')
```

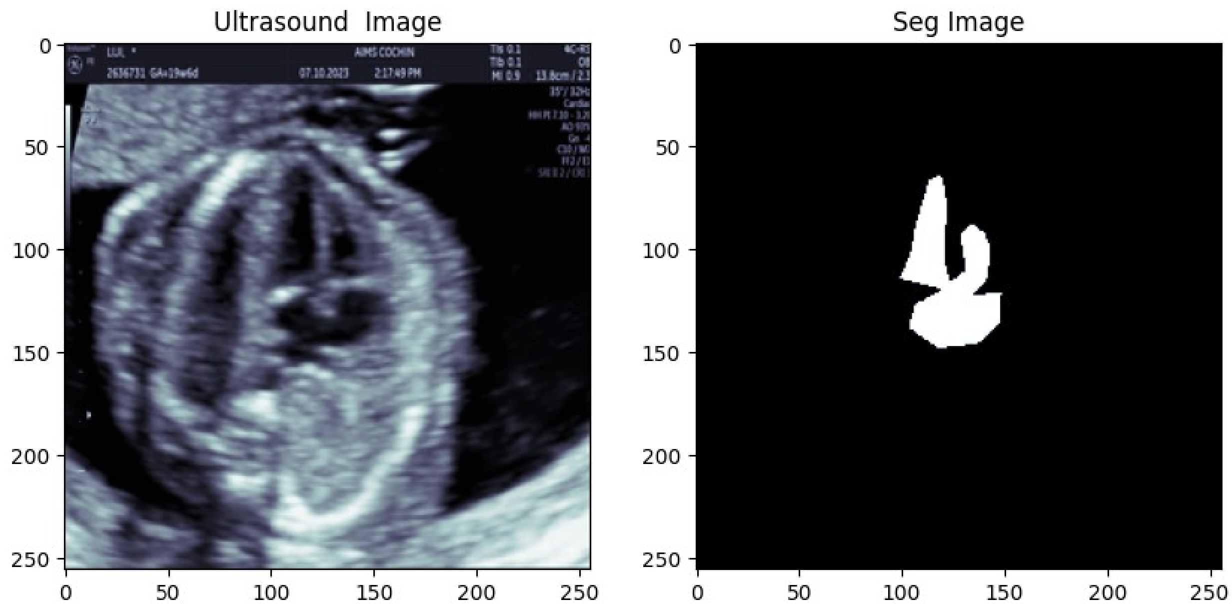
```
ax2.imshow(montage2d(train_Y[:, :, :, 0]), cmap = 'bone')
ax2.set_title('Seg Image')
```

```
Text(0.5, 1.0, 'Seg Image')
```




```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 5))
ax1.imshow(montage2d(test_X[:, :, :, 0]), cmap = 'bone')
ax1.set_title('Ultrasound  Image')
ax2.imshow(montage2d(test_Y[:, :, :, 0]), cmap = 'bone')
ax2.set_title('Seg Image')
```

```
Text(0.5, 1.0, 'Seg Image')
```




```
import numpy as np
import tensorflow as tf

# Define the input shape
IMG_WIDTH = 256
IMG_HEIGHT = 256
OUTPUT_CHANNELS = 3

BUFFER_SIZE = 133
# The batch size of 1 produced better results for the U-Net in the
original pix2pix experiment
BATCH_SIZE = 1

import tensorflow as tf

# Convert the input image to single-channel (grayscale)
t_img_single_channel = tf.image.rgb_to_grayscale(t_img)

# Repeat the single-channel image along the channel dimension to make
it three channels
t_img_rgb = tf.tile(t_img_single_channel, [1, 1, 3])

# Ensure the shape matches the expected input shape of the generator
print(t_img_rgb.shape)


(256, 256, 3)

import tensorflow as tf

def downsample(filters, size, apply_batchnorm=True,
apply_dropout=False, dropout_rate=0.2):
```

```
    initializer = tf.random_normal_initializer(0., 0.02)
    result = tf.keras.Sequential()
    result.add(
        tf.keras.layers.Conv2D(filters, size, strides=2,
padding='same',
                               kernel_initializer=initializer,
use_bias=False))
    if apply_batchnorm:
        result.add(tf.keras.layers.BatchNormalization())
    result.add(tf.keras.layers.LeakyReLU())
    if apply_dropout:
        result.add(tf.keras.layers.Dropout(dropout_rate))
    return result


down_model = downsample(3, 4)
# Convert the input tensor to float32
t_img = tf.cast(t_img, tf.float32)
down_result = down_model(tf.expand_dims(t_img, 0))
print(down_result.shape)
```

```
(1, 128, 128, 3)
```

```
def upsample(filters, size, apply_dropout=False, dropout_rate=0.2):
    initializer = tf.random_normal_initializer(0., 0.02)
    result = tf.keras.Sequential()
    result.add(
        tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                        padding='same',

kernel_initializer=initializer,
                                        use_bias=False))
    result.add(tf.keras.layers.BatchNormalization())
    if apply_dropout:
        result.add(tf.keras.layers.Dropout(dropout_rate))
    result.add(tf.keras.layers.ReLU())
    return result

up_model = upsample(3, 4)
up_result = up_model(down_result)
print (up_result.shape)
```

```
(1, 256, 256, 3)
```

```
def Generator(dropout_rate=0.2):
    inputs = tf.keras.layers.Input(shape=[256, 256, 3])
    down_stack = [
        downsample(64, 4, apply_batchnorm=False, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 128, 128, 64)
        downsample(128, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 64, 64, 128)
```

```
        downsample(256, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 32, 32, 256)
        downsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 16, 16, 512)
        downsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 8, 8, 512)
        downsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 4, 4, 512)
        downsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 2, 2, 512)
        downsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 1, 1, 512)
    ]
    up_stack = [
        upsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 2, 2, 1024)
        upsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 4, 4, 1024)
        upsample(512, 4, apply_dropout=True,
dropout_rate=dropout_rate),  # (batch_size, 8, 8, 1024)
        upsample(512, 4),  # (batch_size, 16, 16, 1024)
        upsample(256, 4),  # (batch_size, 32, 32, 512)
        upsample(128, 4),  # (batch_size, 64, 64, 256)
        upsample(64, 4),  # (batch_size, 128, 128, 128)
    ]
    initializer = tf.random_normal_initializer(0., 0.02)
    last = tf.keras.layers.Conv2DTranspose(3, 4,
                                           strides=2,
                                           padding='same',

kernel_initializer=initializer,
                                           activation='tanh')  #
(batch_size, 256, 256, 3)
    x = inputs
    # Downsampling through the model
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)
    skips = reversed(skips[:-1])
    # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = tf.keras.layers.Concatenate()([x, skip])
    x = last(x)
    return tf.keras.Model(inputs=inputs, outputs=x)

# Instantiate the Generator model with dropout rate of 0.2
generator = Generator(dropout_rate=0.2)
```

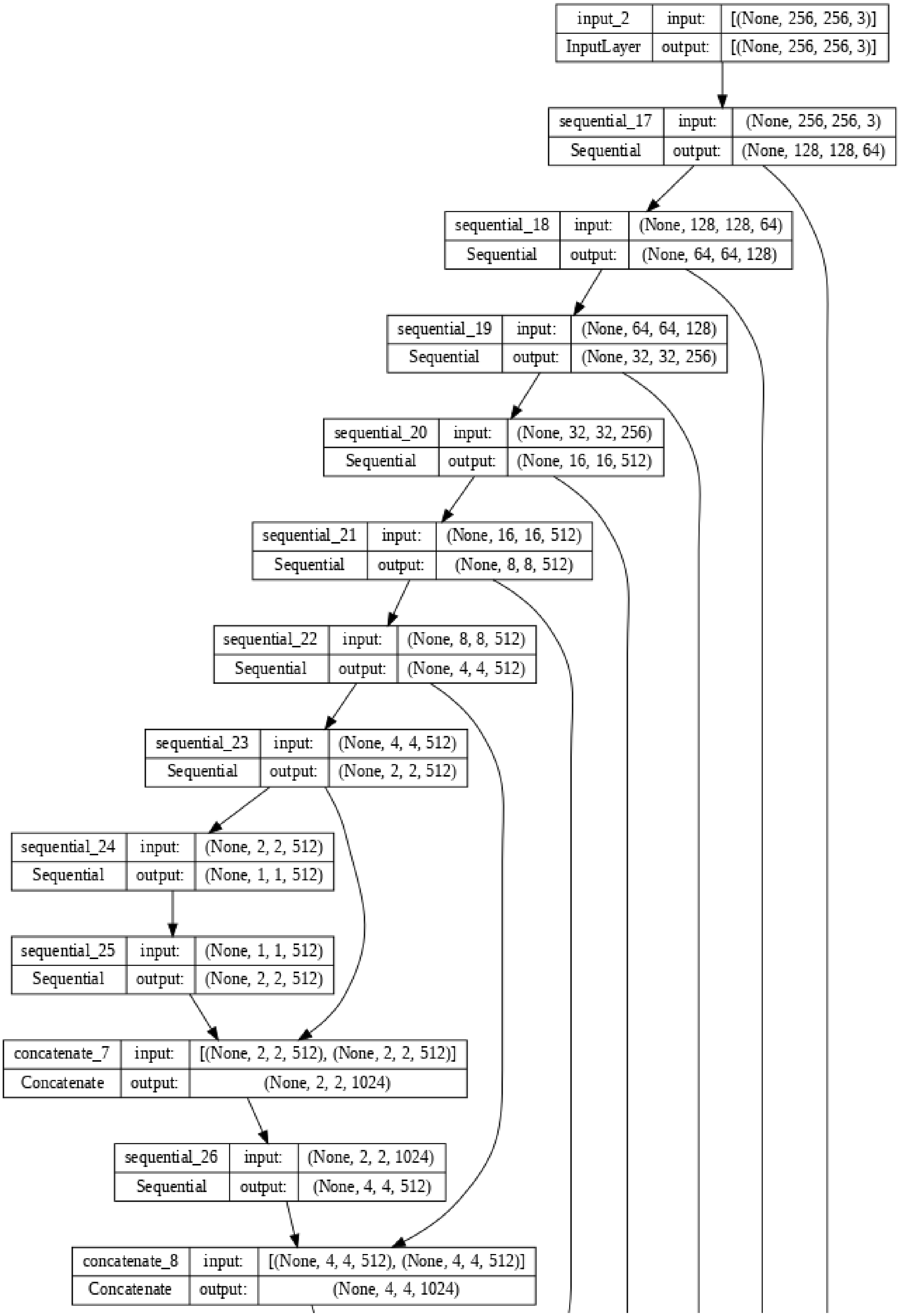
```
generator.compile(loss='binary_crossentropy', optimizer='adam')

# Print the summary of the Generator model
generator.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape                 Param #
Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3)]        0           []


 sequential_2 (Sequential)      (None, 128, 128, 64)         3072
['input_1[0][0]']


 sequential_3 (Sequential)      (None, 64, 64, 128)          131584
['sequential_2[0][0]']


 sequential_4 (Sequential)      (None, 32, 32, 256)          525312
['sequential_3[0][0]']


 sequential_5 (Sequential)      (None, 16, 16, 512)          2099200
['sequential_4[0][0]']


 sequential_6 (Sequential)      (None, 8, 8, 512)            4196352
['sequential_5[0][0]']


 sequential_7 (Sequential)      (None, 4, 4, 512)            4196352
['sequential_6[0][0]']


 sequential_8 (Sequential)      (None, 2, 2, 512)            4196352
['sequential_7[0][0]']


 sequential_9 (Sequential)      (None, 1, 1, 512)            4196352
['sequential_8[0][0]']


 sequential_10 (Sequential)     (None, 2, 2, 512)            4196352
```

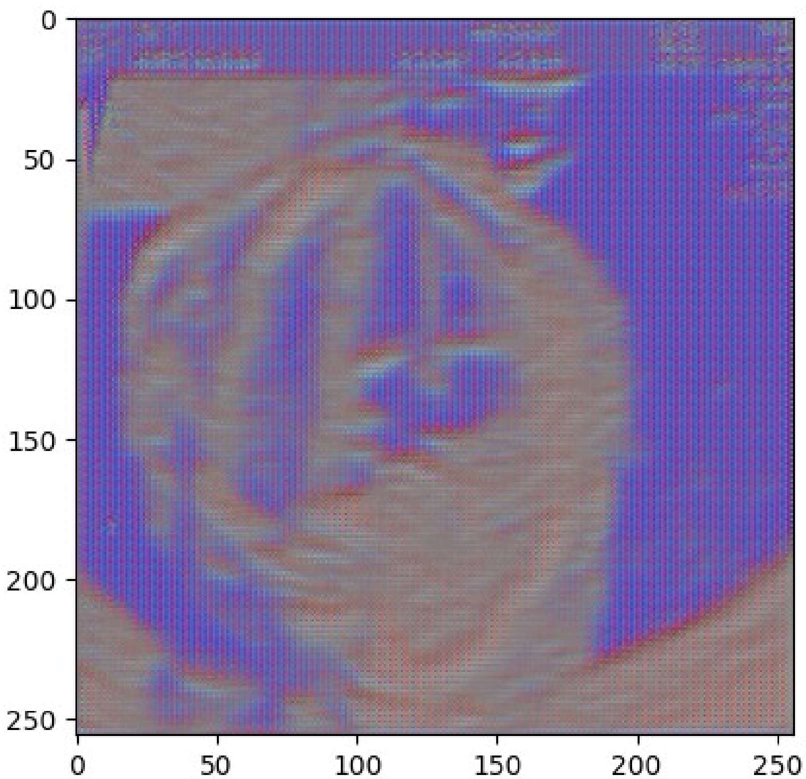
```
['sequential_9[0][0]']


 concatenate (Concatenate)   (None, 2, 2, 1024)           0
['sequential_10[0][0]',

'sequential_8[0][0]']


 sequential_11 (Sequential)  (None, 4, 4, 512)            8390656
['concatenate[0][0]']


 concatenate_1 (Concatenate  (None, 4, 4, 1024)           0
['sequential_11[0][0]',
 )
'sequential_7[0][0]']


 sequential_12 (Sequential)  (None, 8, 8, 512)            8390656
['concatenate_1[0][0]']


 concatenate_2 (Concatenate  (None, 8, 8, 1024)           0
['sequential_12[0][0]',
 )
'sequential_6[0][0]']


 sequential_13 (Sequential)  (None, 16, 16, 512)          8390656
['concatenate_2[0][0]']


 concatenate_3 (Concatenate  (None, 16, 16, 1024)         0
['sequential_13[0][0]',
 )
'sequential_5[0][0]']


 sequential_14 (Sequential)  (None, 32, 32, 256)          4195328
['concatenate_3[0][0]']


 concatenate_4 (Concatenate  (None, 32, 32, 512)          0
['sequential_14[0][0]',
 )
'sequential_4[0][0]']


 sequential_15 (Sequential)  (None, 64, 64, 128)          1049088
```

```
['concatenate_4[0][0]']


 concatenate_5 (Concatenate  (None, 64, 64, 256)          0
['sequential_15[0][0]',
 )
'sequential_3[0][0]']


 sequential_16 (Sequential)  (None, 128, 128, 64)         262400
['concatenate_5[0][0]']


 concatenate_6 (Concatenate  (None, 128, 128, 128)        0
['sequential_16[0][0]',
 )
'sequential_2[0][0]']


 conv2d_transpose_8 (Conv2D  (None, 256, 256, 3)          6147
['concatenate_6[0][0]']
 Transpose)



======================================================================
============================
Total params: 54425859 (207.62 MB)
Trainable params: 54414979 (207.58 MB)
Non-trainable params: 10880 (42.50 KB)
______________________________________________________________________
____________________________
```

```
generator = Generator()
tf.keras.utils.plot_model(generator, show_shapes=True, dpi=64)
```


input_2 | input: | [(None, 256, 256, 3)]
InputLayer | output: | [(None, 256, 256, 3)]
sequential_17 | input: | (None, 256, 256, 3)
Sequential | output: | (None, 128, 128, 64)
sequential_18 | input: | (None, 128, 128, 64)
Sequential | output: | (None, 64, 64, 128)
sequential_19 | input: | (None, 64, 64, 128)
Sequential | output: | (None, 32, 32, 256)
sequential_20 | input: | (None, 32, 32, 256)
Sequential | output: | (None, 16, 16, 512)
sequential_21 | input: | (None, 16, 16, 512)
Sequential | output: | (None, 8, 8, 512)
sequential_22 | input: | (None, 8, 8, 512)
Sequential | output: | (None, 4, 4, 512)
sequential_23 | input: | (None, 4, 4, 512)
Sequential | output: | (None, 2, 2, 512)
sequential_24 | input: | (None, 2, 2, 512)
Sequential | output: | (None, 1, 1, 512)
sequential_25 | input: | (None, 1, 1, 512)
Sequential | output: | (None, 2, 2, 512)
concatenate_7 | input: | [(None, 2, 2, 512), (None, 2, 2, 512)]
Concatenate | output: | (None, 2, 2, 1024)
sequential_26 | input: | (None, 2, 2, 1024)
Sequential | output: | (None, 4, 4, 512)
concatenate_8 | input: | [(None, 4, 4, 512), (None, 4, 4, 512)]
Concatenate | output: | (None, 4, 4, 1024)

```
import tensorflow as tf

# Assuming 'inp' represents a sample input image
inp = test_X[0]

# Generate output from the generator
gen_output = generator(inp[tf.newaxis, ...], training=False)

# Rescale pixel values to [0, 1]
gen_output = (gen_output - tf.reduce_min(gen_output)) /
(tf.reduce_max(gen_output) - tf.reduce_min(gen_output))

# Plot the generated output
plt.imshow(gen_output[0, ...])
plt.show()
```

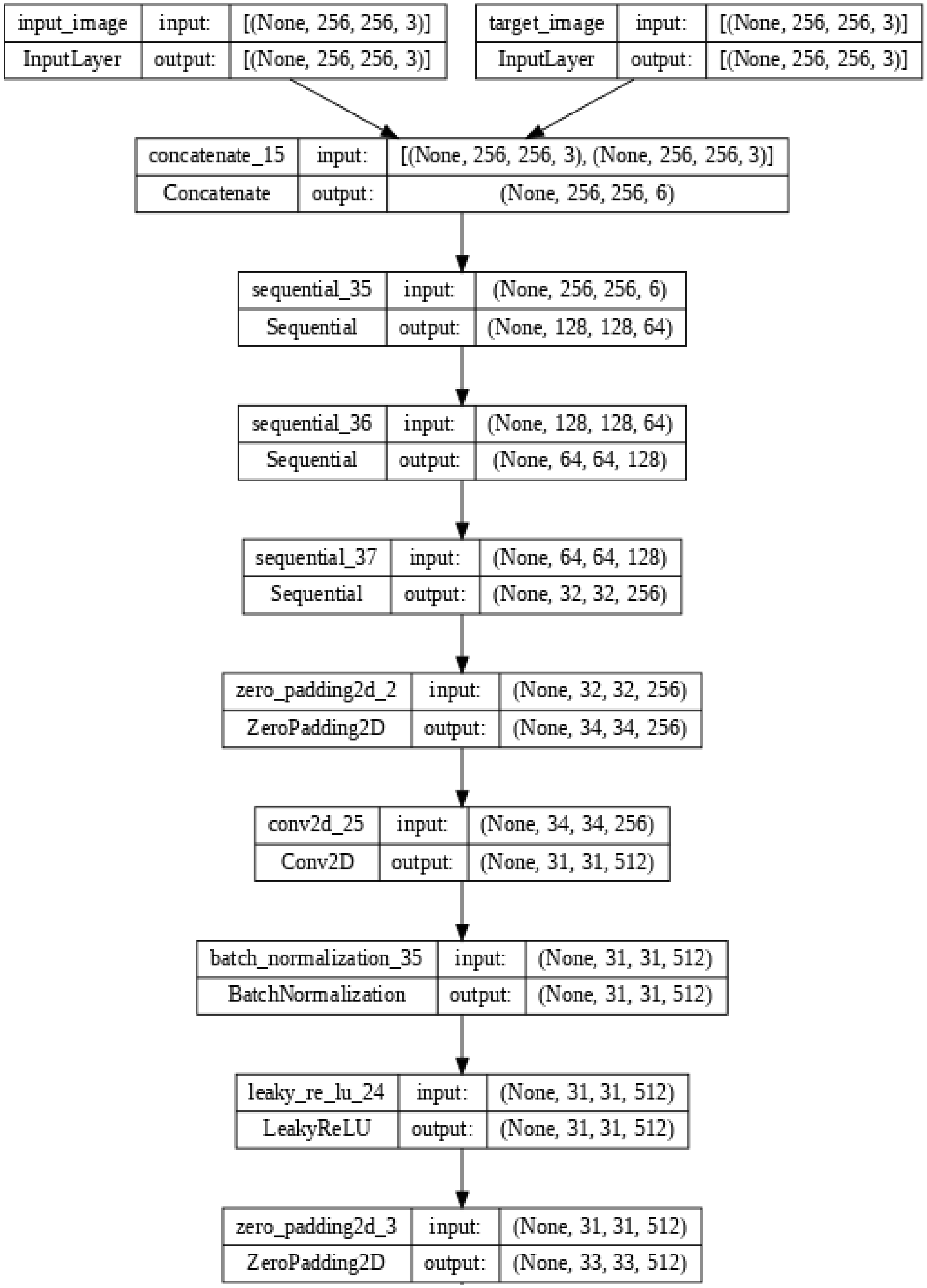



```
LAMBDA = 100

loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(disc_generated_output, gen_output, target):
  gan_loss = loss_object(tf.ones_like(disc_generated_output),
disc_generated_output)

  # Mean absolute error
```

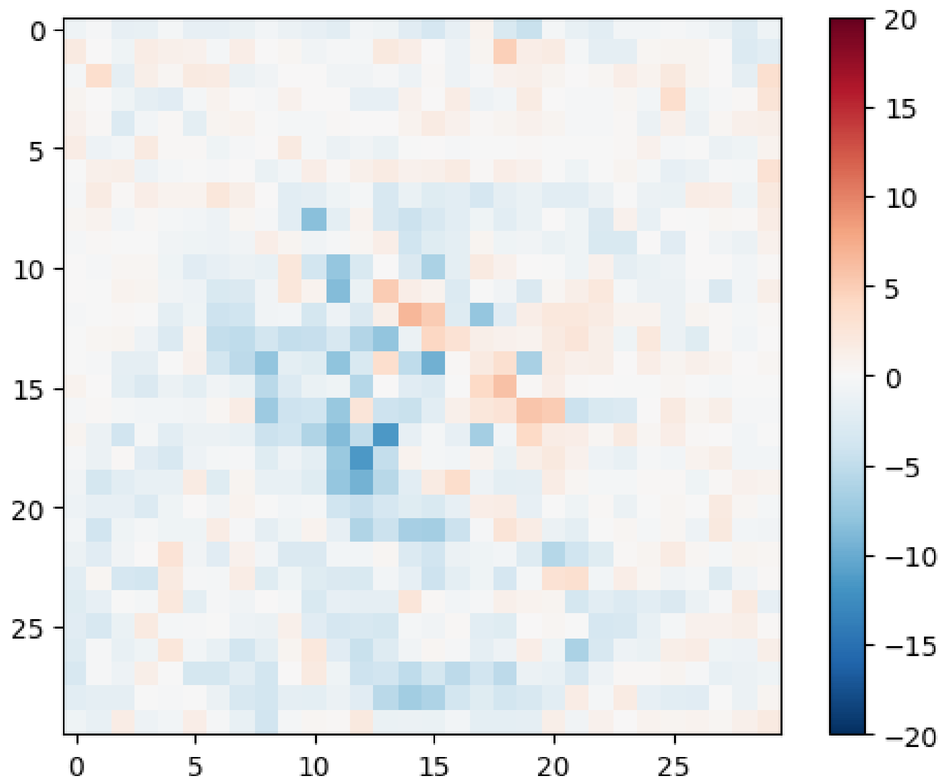
```
  l1_loss = tf.reduce_mean(tf.abs(target - gen_output))

  total_gen_loss = gan_loss + (LAMBDA * l1_loss)

  return total_gen_loss, gan_loss, l1_loss

def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)

  inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
  tar = tf.keras.layers.Input(shape=[256, 256, 3],
name='target_image')

  x = tf.keras.layers.concatenate([inp, tar])  # (batch_size, 256,
256, channels*2)

  down1 = downsample(64, 4, False)(x)  # (batch_size, 128, 128, 64)
  down2 = downsample(128, 4)(down1)  # (batch_size, 64, 64, 128)
  down3 = downsample(256, 4)(down2)  # (batch_size, 32, 32, 256)

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)  # (batch_size,
34, 34, 256)
  conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1)  #
(batch_size, 31, 31, 512)

  batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

  leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)  #
(batch_size, 33, 33, 512)

  last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                kernel_initializer=initializer)
(zero_pad2)  # (batch_size, 30, 30, 1)

  return tf.keras.Model(inputs=[inp, tar], outputs=last)




# Instantiate the Generator model
discriminator = Discriminator()

# Print the summary of the Generator model
discriminator.summary()


discriminator.compile(loss='binary_crossentropy', optimizer='adam')
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_image (InputLayer)       [(None, 256, 256, 3)]  0          []


 target_image (InputLayer)      [(None, 256, 256, 3)]  0          []


 concatenate_14 (Concatenat    (None, 256, 256, 6)  0
['input_image[0][0]',
 e)
'target_image[0][0]']

 sequential_32 (Sequential)    (None, 128, 128, 64)  6144
['concatenate_14[0][0]']

 sequential_33 (Sequential)    (None, 64, 64, 128)  131584
['sequential_32[0][0]']

 sequential_34 (Sequential)    (None, 32, 32, 256)  525312
['sequential_33[0][0]']

 zero_padding2d (ZeroPaddin    (None, 34, 34, 256)  0
['sequential_34[0][0]']
 g2D)


 conv2d_20 (Conv2D)            (None, 31, 31, 512)  2097152
['zero_padding2d[0][0]']

 batch_normalization_32 (Ba    (None, 31, 31, 512)  2048
['conv2d_20[0][0]']
 tchNormalization)


 leaky_re_lu_20 (LeakyReLU)    (None, 31, 31, 512)  0
['batch_normalization_32[0][0]
```

```
                                                                    ']



 zero_padding2d_1 (ZeroPadd  (None, 33, 33, 512)          0
['leaky_re_lu_20[0][0]']
 ing2D)



 conv2d_21 (Conv2D)          (None, 30, 30, 1)            8193
['zero_padding2d_1[0][0]']


======================================================================
============================
Total params: 2770433 (10.57 MB)
Trainable params: 2768641 (10.56 MB)
Non-trainable params: 1792 (7.00 KB)
______________________________________________________________________
____________________________

discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=True, dpi=64)
```


input_image
InputLayer
input:
output:
[(None, 256, 256, 3)]
[(None, 256, 256, 3)]
target_image
InputLayer
input:
output:
[(None, 256, 256, 3)]
[(None, 256, 256, 3)]
concatenate_15
Concatenate
input:
output:
[(None, 256, 256, 3), (None, 256, 256, 3)]
(None, 256, 256, 6)
sequential_35
Sequential
input:
output:
(None, 256, 256, 6)
(None, 128, 128, 64)
sequential_36
Sequential
input:
output:
(None, 128, 128, 64)
(None, 64, 64, 128)
sequential_37
Sequential
input:
output:
(None, 64, 64, 128)
(None, 32, 32, 256)
zero_padding2d_2
ZeroPadding2D
input:
output:
(None, 32, 32, 256)
(None, 34, 34, 256)
conv2d_25
Conv2D
input:
output:
(None, 34, 34, 256)
(None, 31, 31, 512)
batch_normalization_35
BatchNormalization
input:
output:
(None, 31, 31, 512)
(None, 31, 31, 512)
leaky_re_lu_24
LeakyReLU
input:
output:
(None, 31, 31, 512)
(None, 31, 31, 512)
zero_padding2d_3
ZeroPadding2D
input:
output:
(None, 31, 31, 512)
(None, 33, 33, 512)

```
disc_out = discriminator([t_img[tf.newaxis, ...],
m_img[tf.newaxis, ...]], training=False)
plt.imshow(disc_out[0, ..., -1], vmin=-20, vmax=20, cmap='RdBu_r')
plt.colorbar()

<matplotlib.colorbar.Colorbar at 0x7cb37bd07bb0>
```

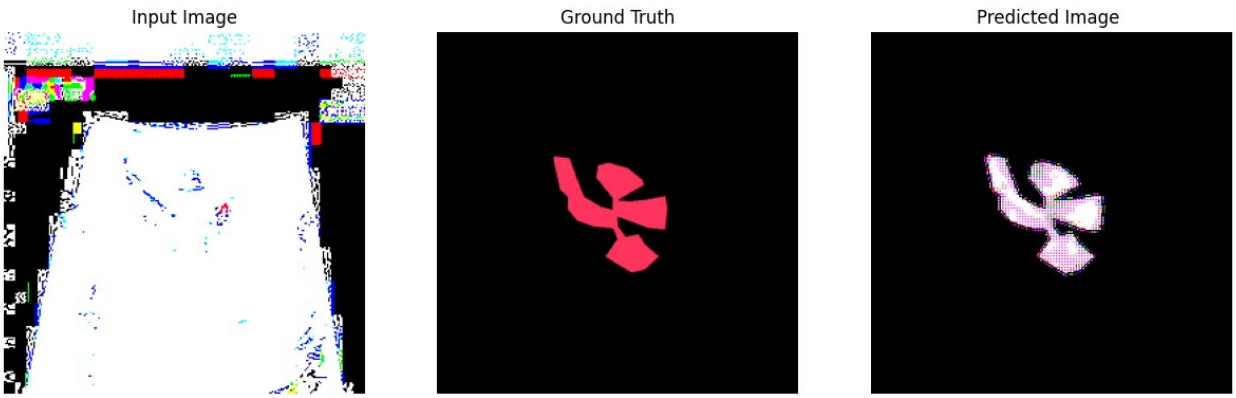



```
def discriminator_loss(disc_real_output, disc_generated_output):
  real_loss = loss_object(tf.ones_like(disc_real_output),
disc_real_output)

  generated_loss = loss_object(tf.zeros_like(disc_generated_output),
disc_generated_output)

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss

generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint =
tf.train.Checkpoint(generator_optimizer=generator_optimizer,
```

```
discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)

def generate_images(model, test_input, tar, metric=None, show=False,
return_prediction=False):
    # Generate prediction using the provided model
    prediction = model(test_input, training=True)

    # Adjust brightness of the predicted image
    prediction = prediction * 0.5 + 0.5

    # Apply ground truth mask to the predicted image
    masked_prediction = prediction * tar

    if show:
        # Plotting the images if show is True
        plt.figure(figsize=(15, 15))
        display_list = [test_input[0], tar[0], masked_prediction[0]]
# Using masked prediction
        title = ['Input Image', 'Ground Truth', 'Predicted Image ']
        for i in range(3):
            plt.subplot(1, 3, i+1)
            plt.title(title[i])
            plt.imshow(display_list[i])
            plt.axis('off')
        plt.show()

    if metric is not None:
        # Update the provided metric if not None
        metric.update_state(tar, prediction)

    if return_prediction:
        # Return prediction if specified
        return prediction

# Assuming generator is the model from code 1
generator = Generator()

# Load the weights of the generator from the checkpoint
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))

# Generate images using the modified function
generate_images(generator, t_img[tf.newaxis, ...],
m_img[tf.newaxis, ...], show=True)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



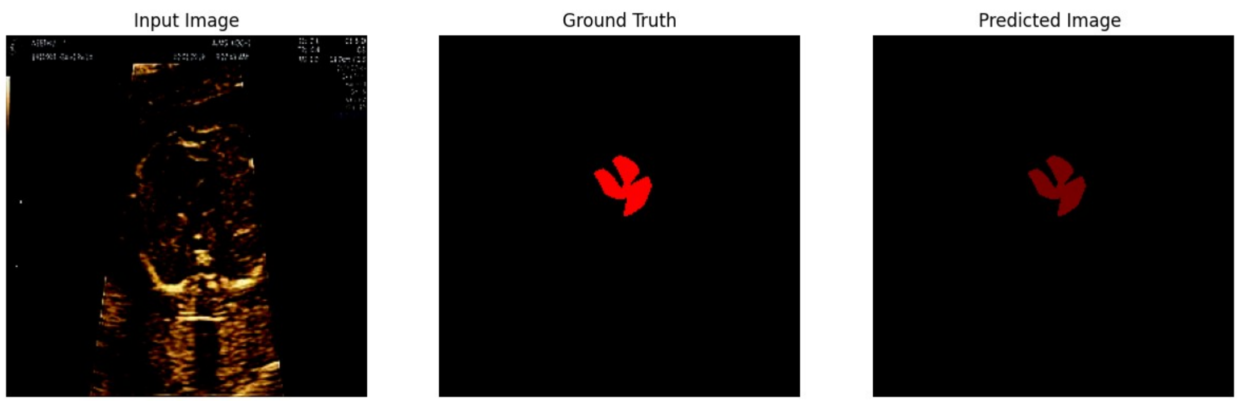

```
import tensorflow as tf

# Define the loss functions
def dice_loss(y_true, y_pred):
    numerator = 2 * tf.reduce_sum(y_true * y_pred)
    denominator = tf.reduce_sum(y_true + y_pred)
    return 1 - (numerator + 1) / (denominator + 1)

def focal_loss(y_true, y_pred, gamma=2.0, alpha=0.25):
    y_pred = tf.clip_by_value(y_pred, 1e-7, 1.0 - 1e-7)
    p_t = tf.where(tf.equal(y_true, 1), y_pred, 1 - y_pred)
    modulating_factor = tf.pow(1.0 - p_t, gamma)
    cross_entropy = -alpha * tf.pow(1.0 - p_t, gamma) *
tf.math.log(p_t)
    return tf.reduce_mean(cross_entropy)

def tversky_loss(y_true, y_pred, alpha=0.7, beta=0.3, smooth=1e-7):
    numerator = tf.reduce_sum(y_true * y_pred, axis=-1)
    denominator = numerator + alpha * tf.reduce_sum(y_true * (1 -
y_pred), axis=-1) + beta * tf.reduce_sum((1 - y_true) * y_pred, axis=-
1)
    return 1 - (numerator + smooth) / (denominator + smooth)

# Define the loss function with class weights
def weighted_loss(y_true, y_pred, class_weights):
    return tf.reduce_mean(class_weights *
tf.keras.losses.binary_crossentropy(y_true, y_pred))

import datetime

log_dir = "logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
```

```
%S"))


# Update the train_step function to use the desired loss function
@tf.function
def train_step(input_image, target, step, metric):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as
disc_tape:
        gen_output = generator(input_image, training=True)
        disc_real_output = discriminator([input_image, target],
training=True)
        disc_generated_output = discriminator([input_image,
gen_output], training=True)

        gen_total_loss, gen_gan_loss, gen_l1_loss =
generator_loss(disc_generated_output, gen_output, target)
        disc_loss = discriminator_loss(disc_real_output,
disc_generated_output)

        # Use the desired loss function here
        gen_loss = dice_loss(target, gen_output)

    generator_gradients = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(generator_gradients,
generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(discriminator_gradients,
discriminator.trainable_variables))

    metric.update_state(target, gen_output)

    with summary_writer.as_default():
        tf.summary.scalar('gen_total_loss', gen_total_loss,
step=step//10)
        tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=step//10)
        tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=step//10)
        tf.summary.scalar('disc_loss', disc_loss, step=step//10)

train_accuracies = []
test_accuracies = []
validation_accuracies = []

import tensorflow as tf

def fit(train_ds, val_ds, steps, class_weights=None):  # Update the
fit function to accept class_weights
    example_input, example_target = next(iter(val_ds))
```

```
    train_metric = tf.keras.metrics.BinaryAccuracy()
    val_metric = tf.keras.metrics.BinaryAccuracy()
    start = time.time()
    step = 0
    for input_image, target in train_ds:
        if (step) % 10 == 0:
            display.clear_output(wait=True)
            train_accuracies.append(train_metric.result().numpy())
            if step != 0:
                print(f'Time taken for 10 steps: {time.time()-
start:.2f} sec\n')

            start = time.time()

            # Validation step
            generate_images(generator, example_input, example_target,
metric=None, show=True)
            val_count = 0
            for val_img, val_target in val_ds:
                generate_images(generator, val_img, val_target,
val_metric)
                val_count += 1
                if val_count >= 28:
                    break
            validation_accuracies.append(val_metric.result().numpy())
            print(f"Step: {step}")
            print(f"Training Accuracy:
{train_metric.result().numpy()}")
            print(f"Validation Accuracy:
{val_metric.result().numpy()}")
            train_metric.reset_states()
            val_metric.reset_states()
            if validation_accuracies[-1] ==
max(validation_accuracies):
                checkpoint.save(file_prefix=checkpoint_prefix)

        # Pass class_weights to the train_step function only if
provided
        if class_weights is not None:
            train_step(input_image, target, step, train_metric,
class_weights)
        else:
            train_step(input_image, target, step, train_metric)

        # Training step
        if (step + 1) % 10 == 0:
            print('.', end='', flush=True)

        step += 1
        if step == steps:
```

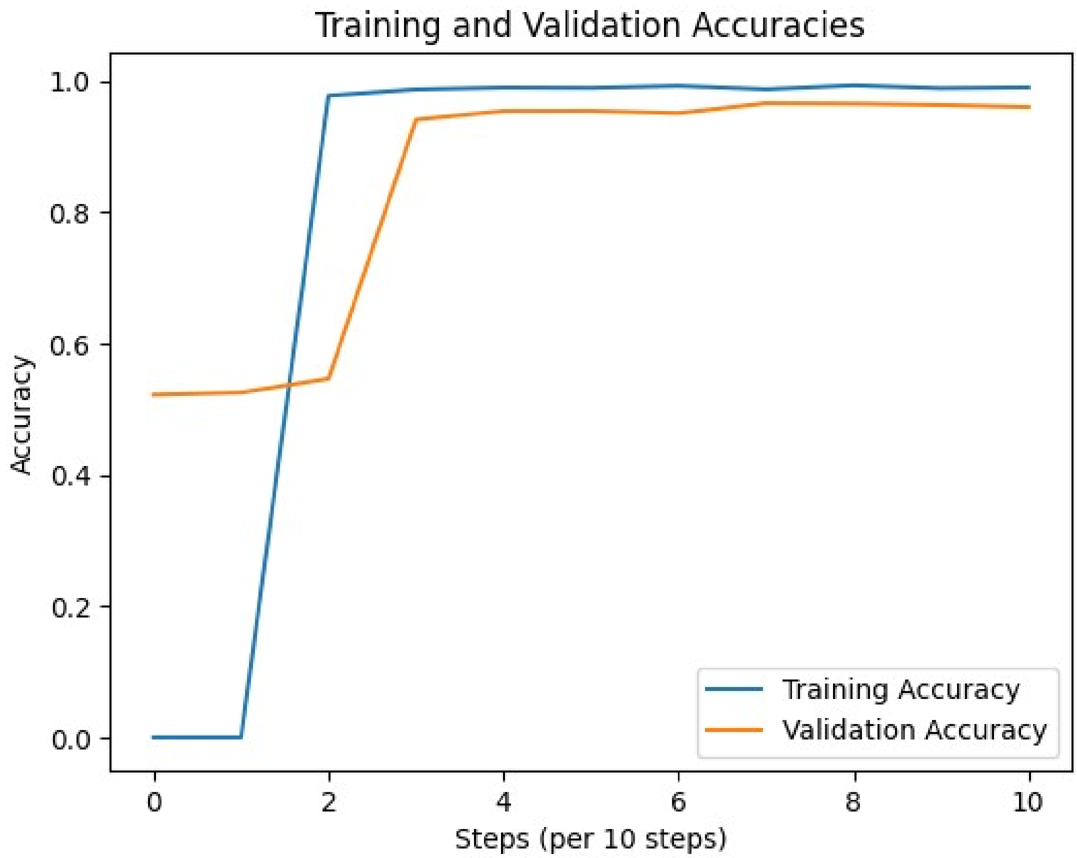
```
            break

# Assuming train_gen and test_gen are the training and testing data
generators respectively
fit(train_generator, test_generator, steps=100)
```

```
Time taken for 10 steps: 82.74 sec


WARNING:matplotlib.image:Clipping input data to the valid range for
imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




```
Step: 90
Training Accuracy: 0.9899856448173523
Validation Accuracy: 0.9601934552192688
.
```

```
import matplotlib.pyplot as plt

# Plotting training and validation accuracies
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(validation_accuracies, label='Validation Accuracy')
plt.xlabel('Steps (per 10 steps)')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracies')
plt.legend()
plt.grid(False)  # Turning off grid
plt.show()
```



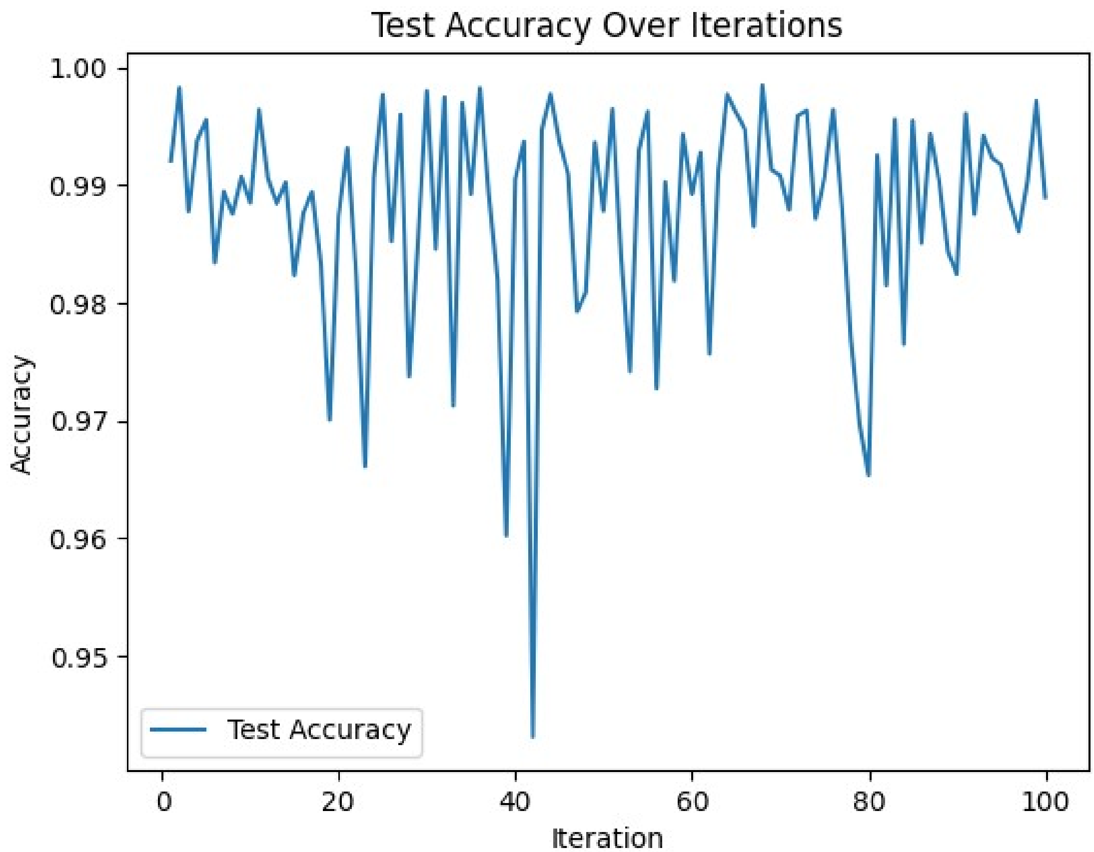

```
import matplotlib.pyplot as plt
import numpy as np

# Define a function to calculate the test accuracy for a subset of the
test dataset
def calculate_test_accuracy(test_ds, num_samples=100):
    test_metric = tf.keras.metrics.BinaryAccuracy()
    count = 0
    for test_input, test_target in test_ds:
        predictions = generator(test_input, training=False)
        test_metric.update_state(test_target, predictions)
        count += 1
        if count >= num_samples:
            break
    test_accuracy = test_metric.result().numpy()
    return test_accuracy

# Load the saved checkpoint
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))

# Define the number of test samples to use for evaluation
num_test_samples = 100
```

```
# Initialize the test accuracy list to store accuracies at each
iteration
test_accuracies = []

# Pass a subset of the test dataset through the generator to generate
images
for i, (test_input, test_target) in enumerate(test_gen):

    # Calculate and print the test accuracy at each iteration
    test_accuracy = calculate_test_accuracy([(test_input,
test_target)], num_samples=1)
    print("Test Accuracy at iteration", i+1, ":", test_accuracy)
    test_accuracies.append(test_accuracy)

    # Break the loop if the desired number of test samples is reached
    if i >= num_test_samples - 1:
        break

# Plot the accuracy graph
plt.figure()
plt.plot(np.arange(1, len(test_accuracies) + 1), test_accuracies,
label='Test Accuracy')
plt.xlabel('Iteration')
plt.ylabel('Accuracy')
plt.title('Test Accuracy Over Iterations')
plt.legend()
plt.show()
```

```
Test Accuracy at iteration 1 : 0.99204504
Test Accuracy at iteration 2 : 0.99825543
Test Accuracy at iteration 3 : 0.9877523
Test Accuracy at iteration 4 : 0.99377954
Test Accuracy at iteration 5 : 0.99553937
Test Accuracy at iteration 6 : 0.98339844
Test Accuracy at iteration 7 : 0.9894613
Test Accuracy at iteration 8 : 0.98753864
Test Accuracy at iteration 9 : 0.9907481
Test Accuracy at iteration 10 : 0.9884898
Test Accuracy at iteration 11 : 0.99642944
Test Accuracy at iteration 12 : 0.9905497
Test Accuracy at iteration 13 : 0.98843384
Test Accuracy at iteration 14 : 0.99022925
Test Accuracy at iteration 15 : 0.9823303
Test Accuracy at iteration 16 : 0.9875996
Test Accuracy at iteration 17 : 0.98942566
Test Accuracy at iteration 18 : 0.98316956
Test Accuracy at iteration 19 : 0.97006726
Test Accuracy at iteration 20 : 0.9873403
Test Accuracy at iteration 21 : 0.99316406
```

```
Test Accuracy at iteration 22 : 0.98236084
Test Accuracy at iteration 23 : 0.9661001
Test Accuracy at iteration 24 : 0.9906209
Test Accuracy at iteration 25 : 0.9976959
Test Accuracy at iteration 26 : 0.9852396
Test Accuracy at iteration 27 : 0.9960073
Test Accuracy at iteration 28 : 0.9737295
Test Accuracy at iteration 29 : 0.9854635
Test Accuracy at iteration 30 : 0.9980062
Test Accuracy at iteration 31 : 0.984553
Test Accuracy at iteration 32 : 0.9974823
Test Accuracy at iteration 33 : 0.971227
Test Accuracy at iteration 34 : 0.9969737
Test Accuracy at iteration 35 : 0.98921204
Test Accuracy at iteration 36 : 0.9982351
Test Accuracy at iteration 37 : 0.98933923
Test Accuracy at iteration 38 : 0.98191833
Test Accuracy at iteration 39 : 0.9602407
Test Accuracy at iteration 40 : 0.99048364
Test Accuracy at iteration 41 : 0.9937083
Test Accuracy at iteration 42 : 0.9430949
Test Accuracy at iteration 43 : 0.9946645
Test Accuracy at iteration 44 : 0.99775183
Test Accuracy at iteration 45 : 0.99367774
Test Accuracy at iteration 46 : 0.9909159
Test Accuracy at iteration 47 : 0.97924805
Test Accuracy at iteration 48 : 0.9808604
Test Accuracy at iteration 49 : 0.9936371
Test Accuracy at iteration 50 : 0.98779297
Test Accuracy at iteration 51 : 0.9964752
Test Accuracy at iteration 52 : 0.9838358
Test Accuracy at iteration 53 : 0.97416687
Test Accuracy at iteration 54 : 0.992925
Test Accuracy at iteration 55 : 0.99627686
Test Accuracy at iteration 56 : 0.97273254
Test Accuracy at iteration 57 : 0.99024963
Test Accuracy at iteration 58 : 0.98186237
Test Accuracy at iteration 59 : 0.9943085
Test Accuracy at iteration 60 : 0.9892222
Test Accuracy at iteration 61 : 0.9927572
Test Accuracy at iteration 62 : 0.97566736
Test Accuracy at iteration 63 : 0.99110925
Test Accuracy at iteration 64 : 0.99773663
Test Accuracy at iteration 65 : 0.9961548
Test Accuracy at iteration 66 : 0.9947561
Test Accuracy at iteration 67 : 0.986496
Test Accuracy at iteration 68 : 0.99846905
Test Accuracy at iteration 69 : 0.99129736
Test Accuracy at iteration 70 : 0.9908295
```

```
Test Accuracy at iteration 71 : 0.9878794
Test Accuracy at iteration 72 : 0.9958852
Test Accuracy at iteration 73 : 0.9963583
Test Accuracy at iteration 74 : 0.9871521
Test Accuracy at iteration 75 : 0.9905446
Test Accuracy at iteration 76 : 0.9964142
Test Accuracy at iteration 77 : 0.9880625
Test Accuracy at iteration 78 : 0.9768473
Test Accuracy at iteration 79 : 0.9697011
Test Accuracy at iteration 80 : 0.96533203
Test Accuracy at iteration 81 : 0.99253845
Test Accuracy at iteration 82 : 0.98148096
Test Accuracy at iteration 83 : 0.9955699
Test Accuracy at iteration 84 : 0.97647095
Test Accuracy at iteration 85 : 0.9954732
Test Accuracy at iteration 86 : 0.9850464
Test Accuracy at iteration 87 : 0.994339
Test Accuracy at iteration 88 : 0.99033105
Test Accuracy at iteration 89 : 0.9843241
Test Accuracy at iteration 90 : 0.98244226
Test Accuracy at iteration 91 : 0.9960989
Test Accuracy at iteration 92 : 0.9875081
Test Accuracy at iteration 93 : 0.9941966
Test Accuracy at iteration 94 : 0.992335
Test Accuracy at iteration 95 : 0.9916891
Test Accuracy at iteration 96 : 0.98862714
Test Accuracy at iteration 97 : 0.9860179
Test Accuracy at iteration 98 : 0.990331
Test Accuracy at iteration 99 : 0.99715173
Test Accuracy at iteration 100 : 0.9889577
```



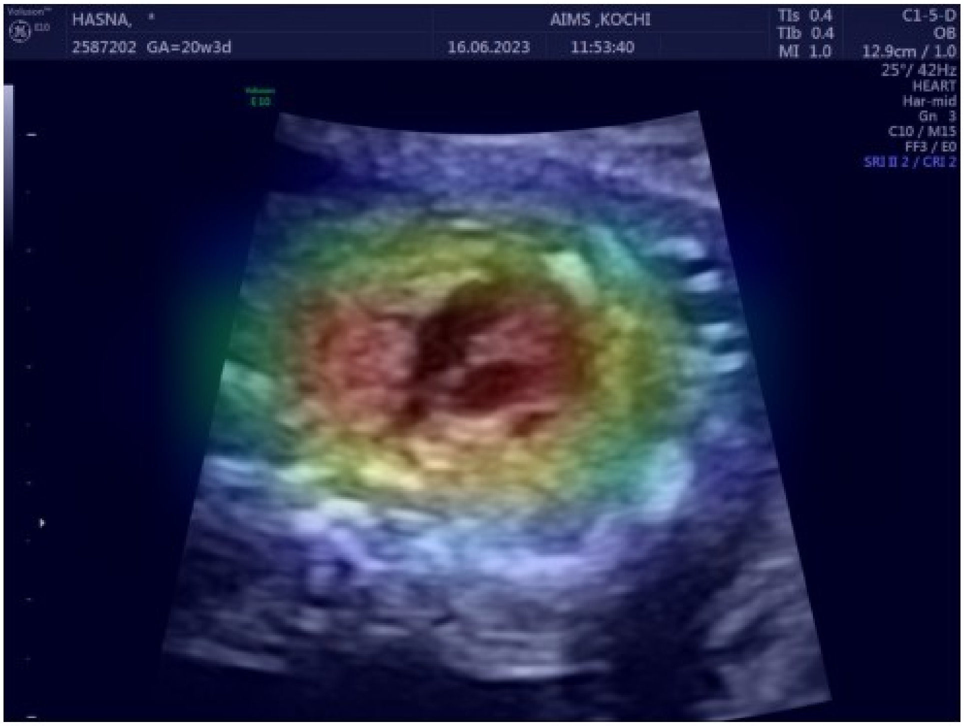
GRAD CAM VISUALIZATION

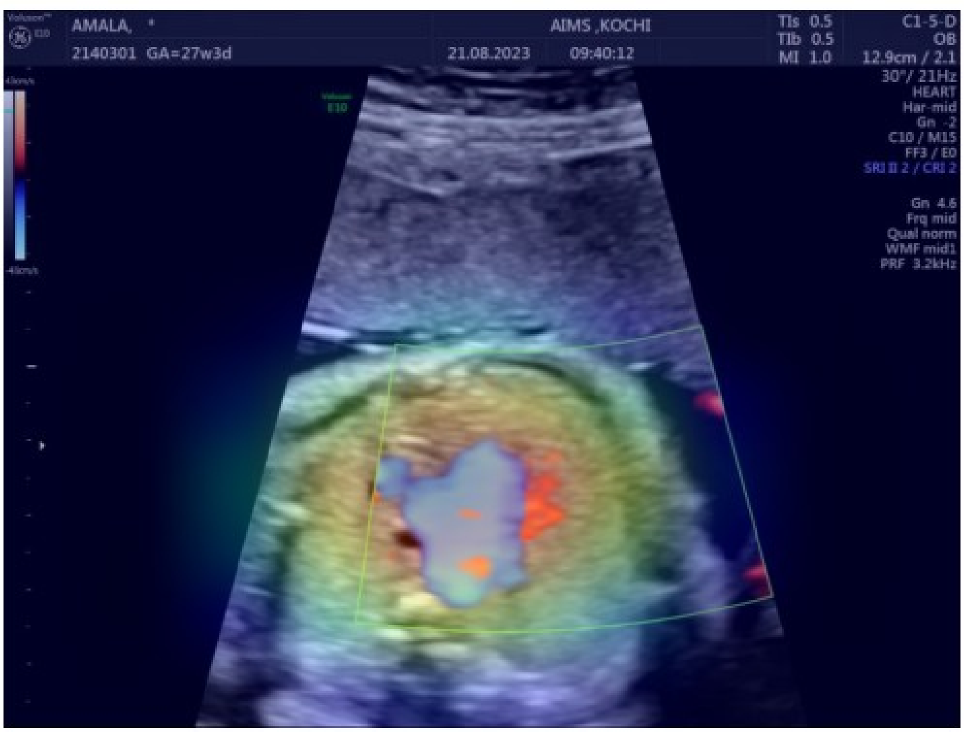
```
import os

os.environ["KERAS_BACKEND"] = "tensorflow"

import numpy as np
import tensorflow as tf
import keras

# Display
from IPython.display import Image, display
import matplotlib as mpl
import matplotlib.pyplot as plt

original_dataset_path = "/content/drive/MyDrive/HEART US IMAGE FETUS"
mask_dataset_path =
"/content/drive/MyDrive/FETUS_ULTRASOUND_IMAGES_HEART_CHAMBERS/Segment
ationClass"

import os
import random
import matplotlib.pyplot as plt
from PIL import Image
```

```
# Function to get random pairs of original image and mask paths from
the dataset
def get_random_img_and_mask_paths(original_dataset_path,
mask_dataset_path, num_images=15):
    all_files = os.listdir(original_dataset_path)
    image_files = [f for f in all_files if f.endswith('.jpg')]
    random_img_paths = random.sample(image_files, min(num_images,
len(image_files)))
    mask_paths = [os.path.join(mask_dataset_path, img.replace('.jpg',
'.png')) for img in random_img_paths]
    return [(os.path.join(original_dataset_path, img), mask) for img,
mask in zip(random_img_paths, mask_paths)]

# Get random pairs of original image and mask paths
random_img_and_mask_paths =
get_random_img_and_mask_paths(original_dataset_path,
mask_dataset_path, num_images=15)

# Define the save_and_display_gradcam function for images and masks
def save_and_display_gradcam_for_images_and_masks(img_and_mask_paths,
model, preprocess_input, get_img_array, make_gradcam_heatmap,
last_conv_layer_name):
    for img_path, mask_path in img_and_mask_paths:
        # Prepare original image
        original_img_array = preprocess_input(get_img_array(img_path,
size=(299, 299)))

        # Prepare mask
        mask_img = Image.open(mask_path).convert('L')  # Convert to
grayscale
        mask_img = mask_img.resize((original_img_array.shape[2],
original_img_array.shape[1]), Image.NEAREST)  # Resize mask to match
image dimensions
        mask_array = np.array(mask_img) / 255.0  # Normalize mask
        mask_array = mask_array.reshape((original_img_array.shape[1],
original_img_array.shape[2], 1))  # Add extra dimension to match
channels

        # Combine original image and mask
        img_with_mask = original_img_array * mask_array

        # Make model
        model = model_builder(weights="imagenet")

        # Remove last layer's softmax
        model.layers[-1].activation = None

        # Generate class activation heatmap
        heatmap = make_gradcam_heatmap(img_with_mask, model,
```

```
last_conv_layer_name)

        # Load the original image
        img = keras.preprocessing.image.load_img(img_path)
        img = keras.preprocessing.image.img_to_array(img)

        # Rescale heatmap to a range 0-255
        heatmap = np.uint8(255 * heatmap)

        # Use jet colormap to colorize heatmap
        jet = plt.cm.get_cmap("jet")

        # Use RGB values of the colormap
        jet_colors = jet(np.arange(256))[:, :3]
        jet_heatmap = jet_colors[heatmap]

        # Create an image with RGB colorized heatmap
        jet_heatmap =
keras.preprocessing.image.array_to_img(jet_heatmap)
        jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
        jet_heatmap =
keras.preprocessing.image.img_to_array(jet_heatmap)

        # Superimpose the heatmap on original image
        superimposed_img = jet_heatmap * 0.4 + img
        superimposed_img =
keras.preprocessing.image.array_to_img(superimposed_img)

        # Display Grad CAM
        plt.figure()
        plt.imshow(superimposed_img)
        plt.axis('off')
        plt.show()

# Use the save_and_display_gradcam_for_images_and_masks function
save_and_display_gradcam_for_images_and_masks(random_img_and_mask_path
s, model_builder, preprocess_input, get_img_array,
make_gradcam_heatmap, last_conv_layer_name)
```

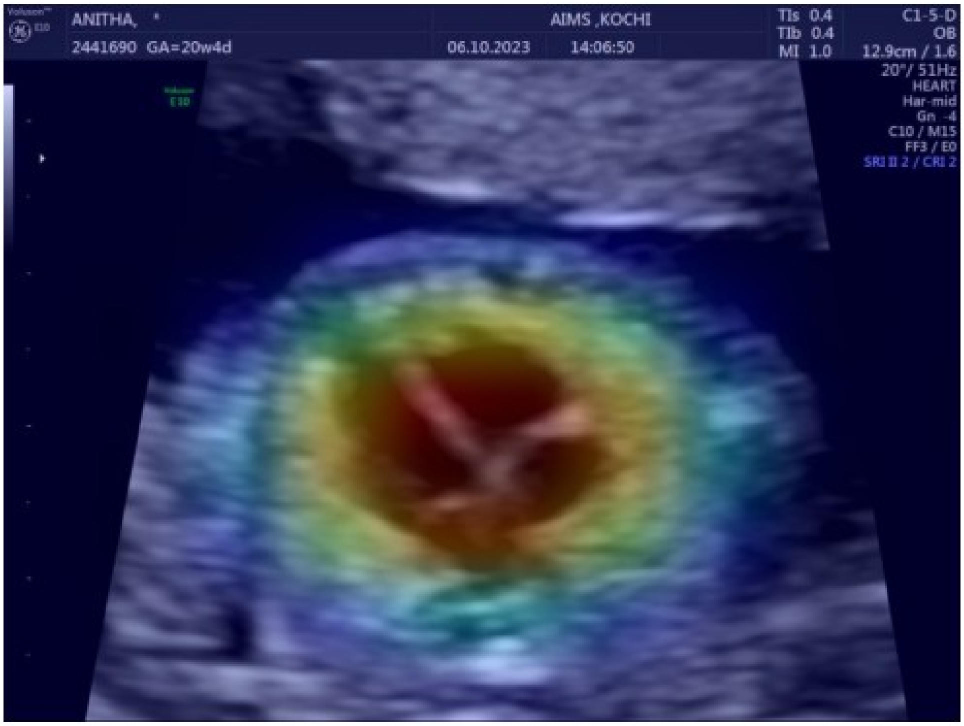
```
<ipython-input-100-9dacdcb4a280>:49: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
two minor releases later. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap(obj)`` instead.
  jet = plt.cm.get_cmap("jet")
```

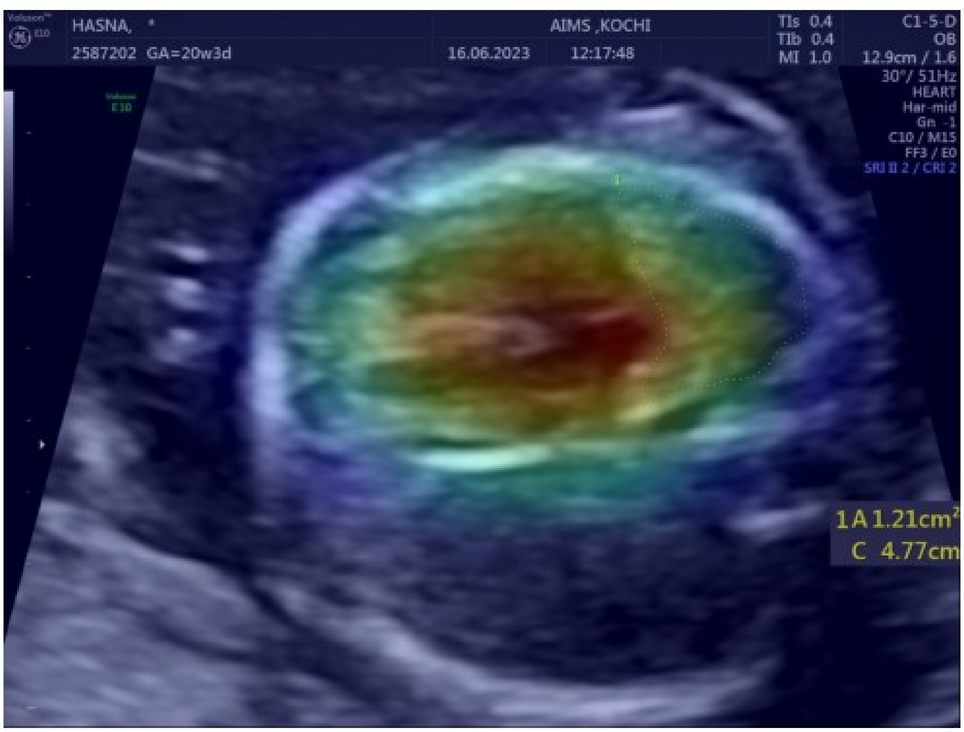

HASNA, *
AIMS ,KOCHI
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
2587202 GA=20w3d
16.06.2023
11:53:40
12.9cm / 1.0
25°/ 42Hz
HEART
Har-mid
Gn 3
C10 / M15
FF3 / E0
SRI II 2 / CRI 2

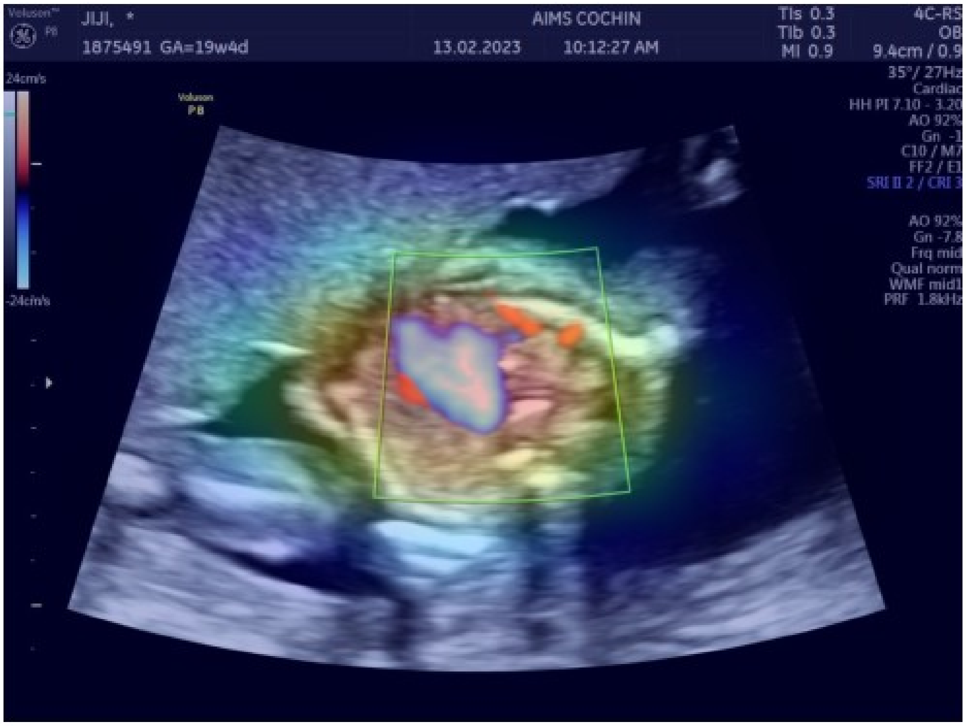

AMALA, *
AIMS ,KOCHI
TIs 0.5
TIb 0.5
MI 1.0
C1-5-D
OB
2140301 GA=27w3d
21.08.2023
09:40:12
12.9cm / 2.1
30°/ 21Hz
HEART
Har-mid
Gn -2
C10 / M15
FF3 / E0
SRI II 2 / CRI 2
Gn 4.6
Frq mid
Qual norm
WMF mid1
PRF 3.2kHz

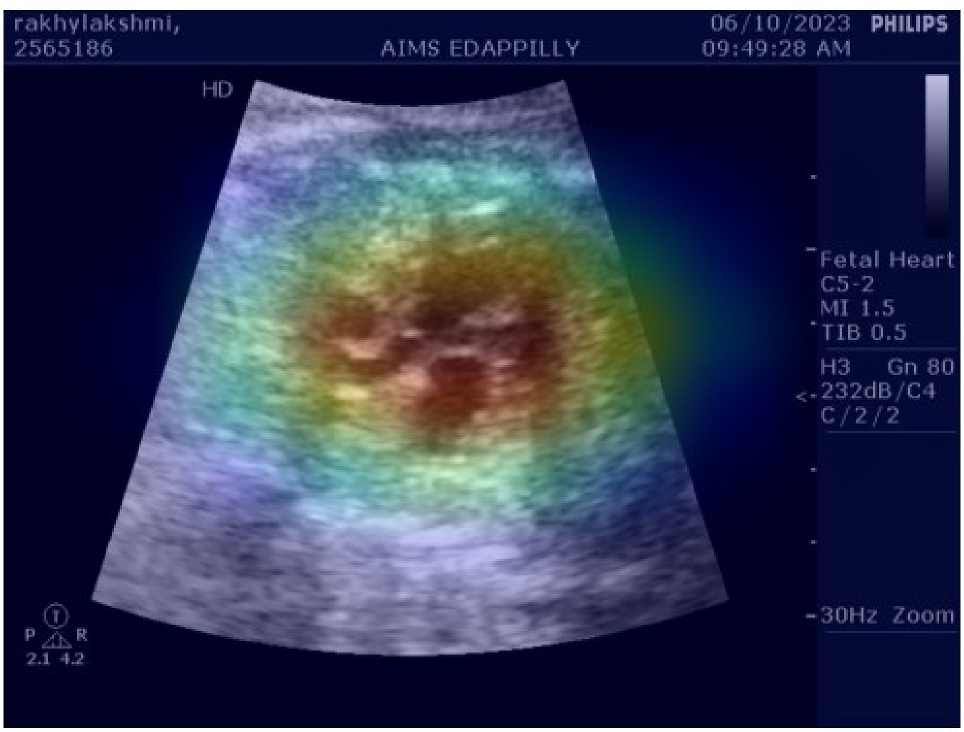

DIVYA, *
2503235 GA=24w3d
AIMS ,KOCHI
05.04.2023
08:34:48
TIs 0.5
TIb 0.5
MI 1.0
C1-5-D
OB
9.6cm / 1.2
28°/ 55Hz
HEART
Har-mid
Gn 2
C10 / M15
FF3 / E0
SRI II 2 / CRI 2
hd

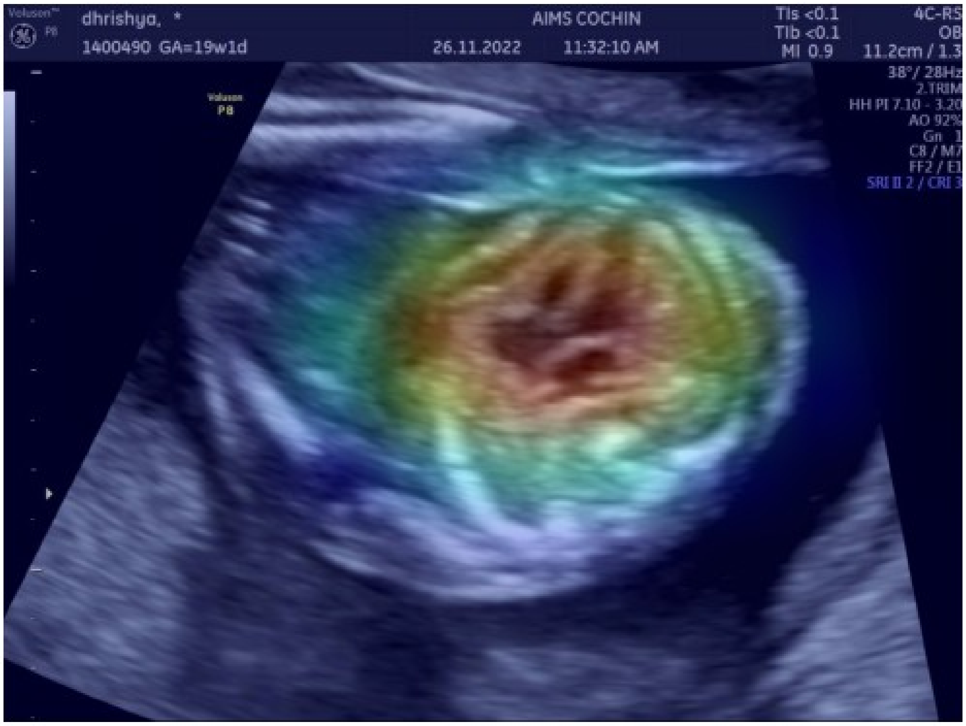

SUSAN, *
2632754 GA=25w5d
AIMS COCHIN
27.09.2023
1:26:35 PM
TIs 0.3
TIb 0.3
MI 0.9
4C-RS
OB
11.5cm / 1.5
35°/ 21Hz
Cardiac
HH PI 7.10 - 3.20
AO 93%
Gn -1
C10 / M7
FF2 / E1
SRI II 2 / CRI 3
AO 93%
Gn -7.8
Frq mid
Qual norm
WMF mid1
PRF 3.2kHz
43cm/s
-43cm/s

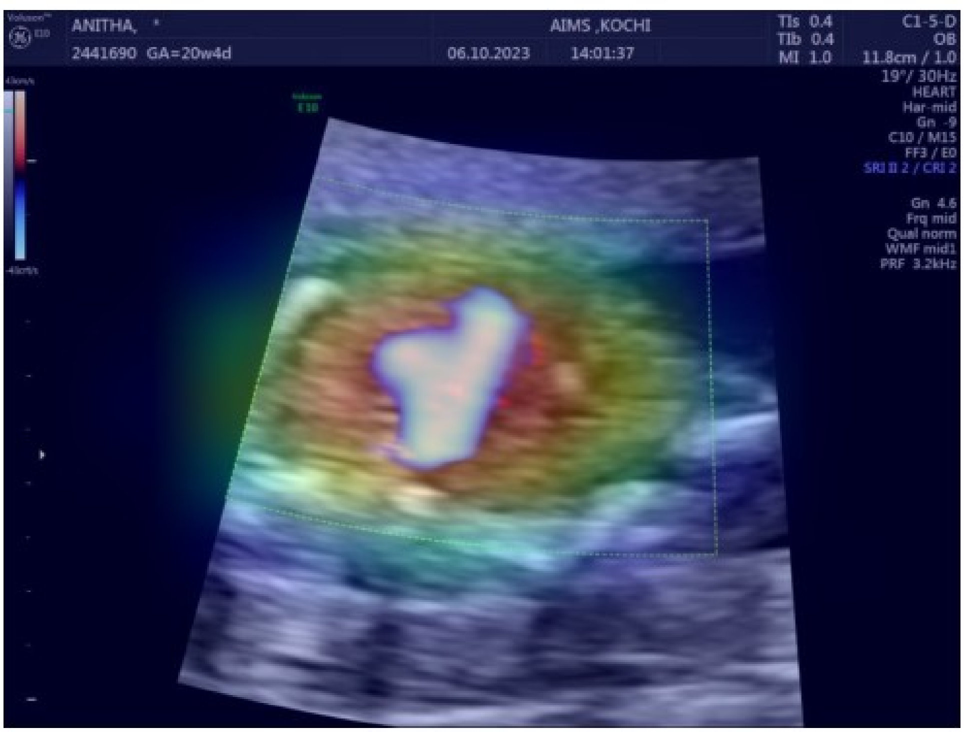

ANITHA, *
AIMS ,KOCHI
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
2441690 GA=20w4d
06.10.2023
14:06:50
12.9cm / 1.6
20°/ 51Hz
HEART
Har-mid
Gn -4
C10 / M15
FF3 / E0
SRI II 2 / CRI 2

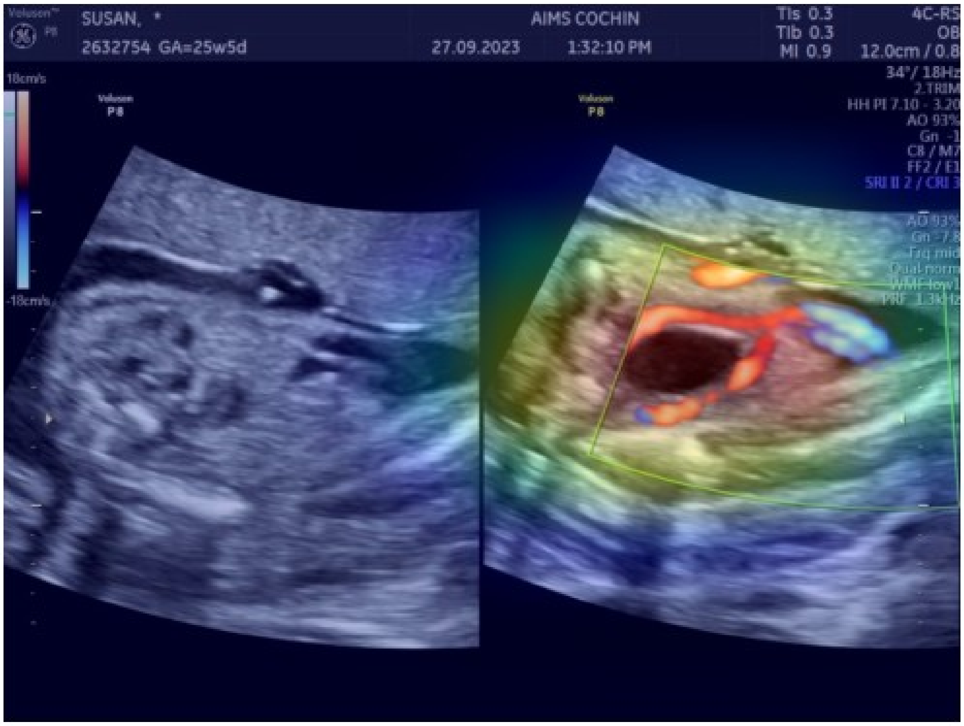

HASNA, *
AIMS ,KOCHI
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
2587202 GA=20w3d
16.06.2023
12:17:48
12.9cm / 1.6
30°/ 51Hz
HEART
Har-mid
Gn -1
C10 / M15
FF3 / E0
SRI II 2 / CRI 2
1 A 1.21cm²
C 4.77cm

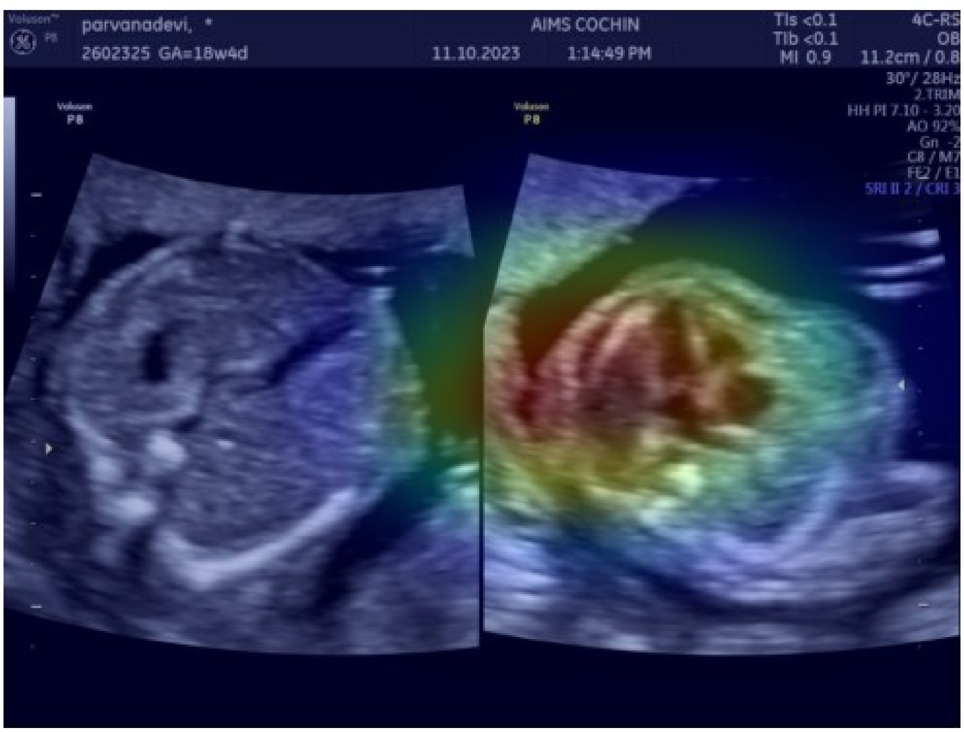

JIJI, *
1875491 GA=19w4d
AIMS COCHIN
13.02.2023
10:12:27 AM
TIs 0.3
TIb 0.3
MI 0.9
4C-RS
OB
9.4cm / 0.9
35°/ 27Hz
Cardiac
AO 92%
Gn -7.8
Frq mid
Qual norm
PRF 1.8kHz
24cm/s
-24cm/s


rakhylakshmi,
2565186
AIMS EDAPPILLY
06/10/2023
09:49:28 AM
PHILIPS
HD
Fetal Heart
C5-2
MI 1.5
TIB 0.5
H3 Gn 80
232dB/C4
C/2/2
30Hz Zoom
P R
2.1 4.2

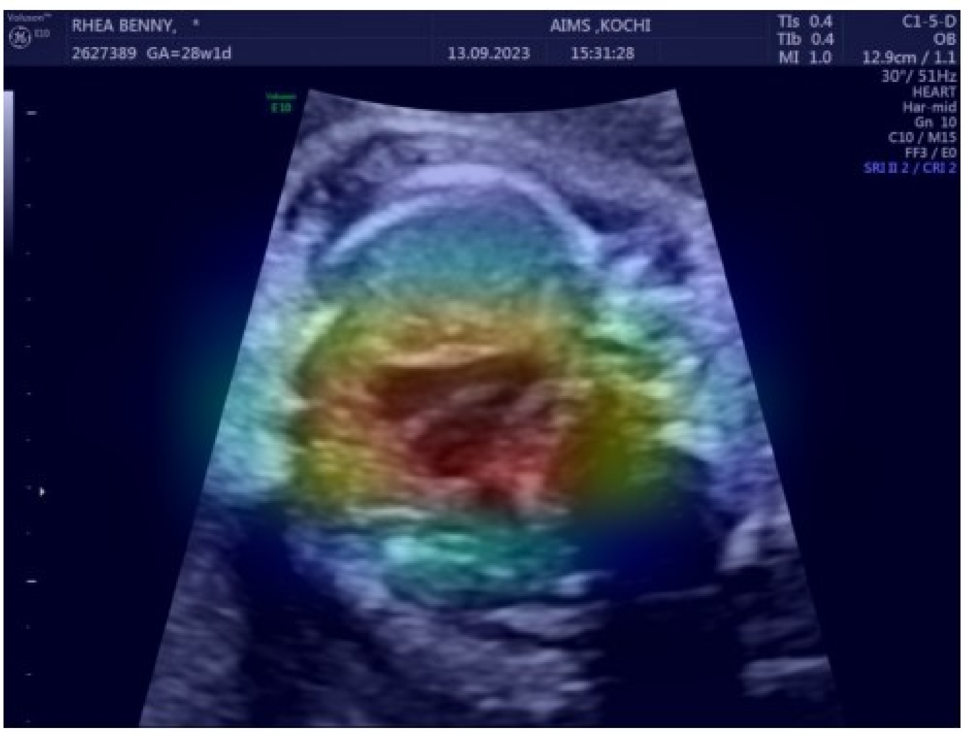

Voluson™
P8
dhrishya, *
1400490 GA=19w1d
AIMS COCHIN
26.11.2022
11:32:10 AM
TIs <0.1
TIb <0.1
MI 0.9
4C-RS
OB
11.2cm / 1.3
38°/ 28Hz
2.TRIM
HH PI 7.10 - 3.20
AO 92%
Gn 1
C8 / M7
FF2 / E1
SRI II 2 / CRI 3
Voluson
P8

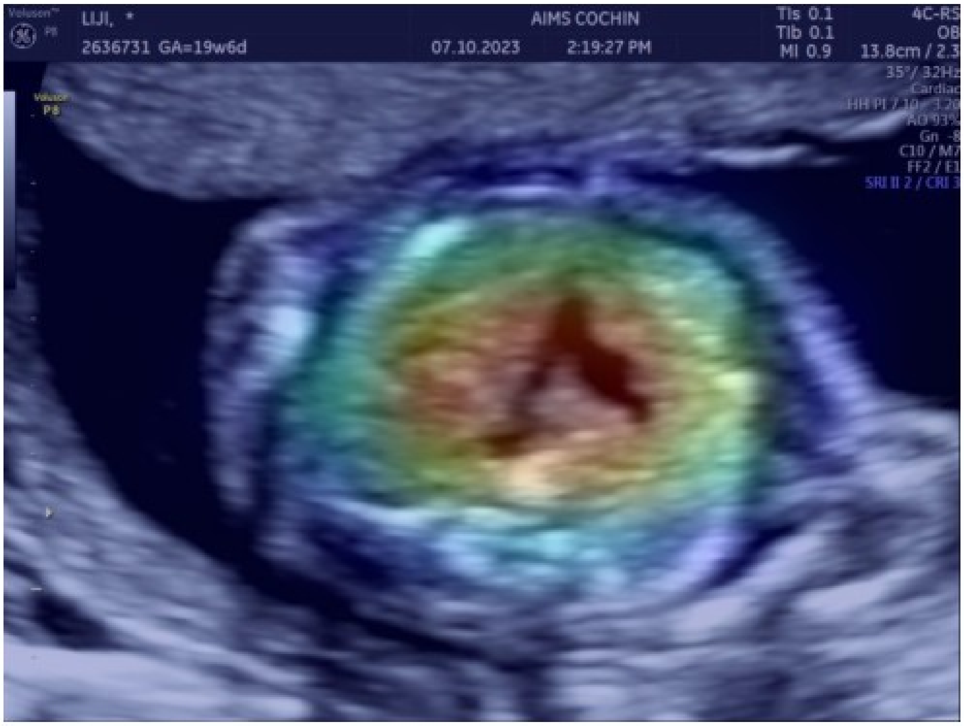

Voluson™
E10
ANITHA, *
2441690 GA=20w4d
AIMS ,KOCHI
06.10.2023
14:01:37
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
11.8cm / 1.0
19°/ 30Hz
HEART
Har-mid
Gn -9
C10 / M15
FF3 / E0
SRI II 2 / CRI 2
Gn 4.6
Frq mid
Qual norm
WMF mid1
PRF 3.2kHz


SUSAN, *
2632754 GA=25w5d
AIMS COCHIN
27.09.2023
1:32:10 PM
TIs 0.3
TIb 0.3
MI 0.9
4C-RS
OB
12.0cm / 0.8
34°/ 18Hz
2.TRIM
HH PI 7.10 - 3.20
AO 93%
18cm/s
-18cm/s


parvanadevi, *
2602325 GA=18w4d
AIMS COCHIN
11.10.2023
1:14:49 PM
TIs <0.1
TIb <0.1
MI 0.9
4C-RS
OB
11.2cm / 0.8
30°/ 28Hz
2.TRIM
HH PI 7.10 - 3.20
AO 92%


HASNA, *
2587202 GA=20w3d
AIMS ,KOCHI
16.06.2023
11:53:29
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
12.9cm / 1.0
25°/ 42Hz
HEART
Har-mid
Gn 3
C10 / M15
FF3 / E0
SRI II 2 / CRI 2


RHEA BENNY, *
2627389 GA=28w1d
AIMS ,KOCHI
13.09.2023
15:31:28
TIs 0.4
TIb 0.4
MI 1.0
C1-5-D
OB
12.9cm / 1.1
30°/ 51Hz
HEART
Har-mid
Gn 10
C10 / M15
FF3 / E0
SRI II 2 / CRI 2


LIJI, *
2636731 GA=19w6d
AIMS COCHIN
07.10.2023
2:19:27 PM
TIs 0.1
TIb 0.1
MI 0.9
4C-RS
OB
13.8cm / 2.3
35°/ 32Hz
Cardiac
Gn -8
C10 / M7
FF2 / E1