Problem 1) Movie Reviews dataset

a)

In [ ]:
```
!pip install gensim
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
# imports

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import seaborn as sns

import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
nltk.download('omw-1.4')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

from gensim.models import Word2Vec

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Bidirec
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, precision

from wordcloud import WordCloud

import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [ ]:

```
# loding the movies IMDB dataset
df = pd.read_csv('/content/drive/MyDrive/IMDB Dataset.csv')
```

In [ ]:

```
# display first 5 rows
df.head(5)
```

Out[ ]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

In [ ]:

```
df.sentiment.value_counts()
```

Out[ ]:

sentiment	count
positive	25000
negative	25000

dtype: int64

In [ ]:

```
df.isna().sum() # checking for null
```

Out[ ]:

	0
review	0
sentiment	0

dtype: int64

In [ ]:

```
df.dtypes
```

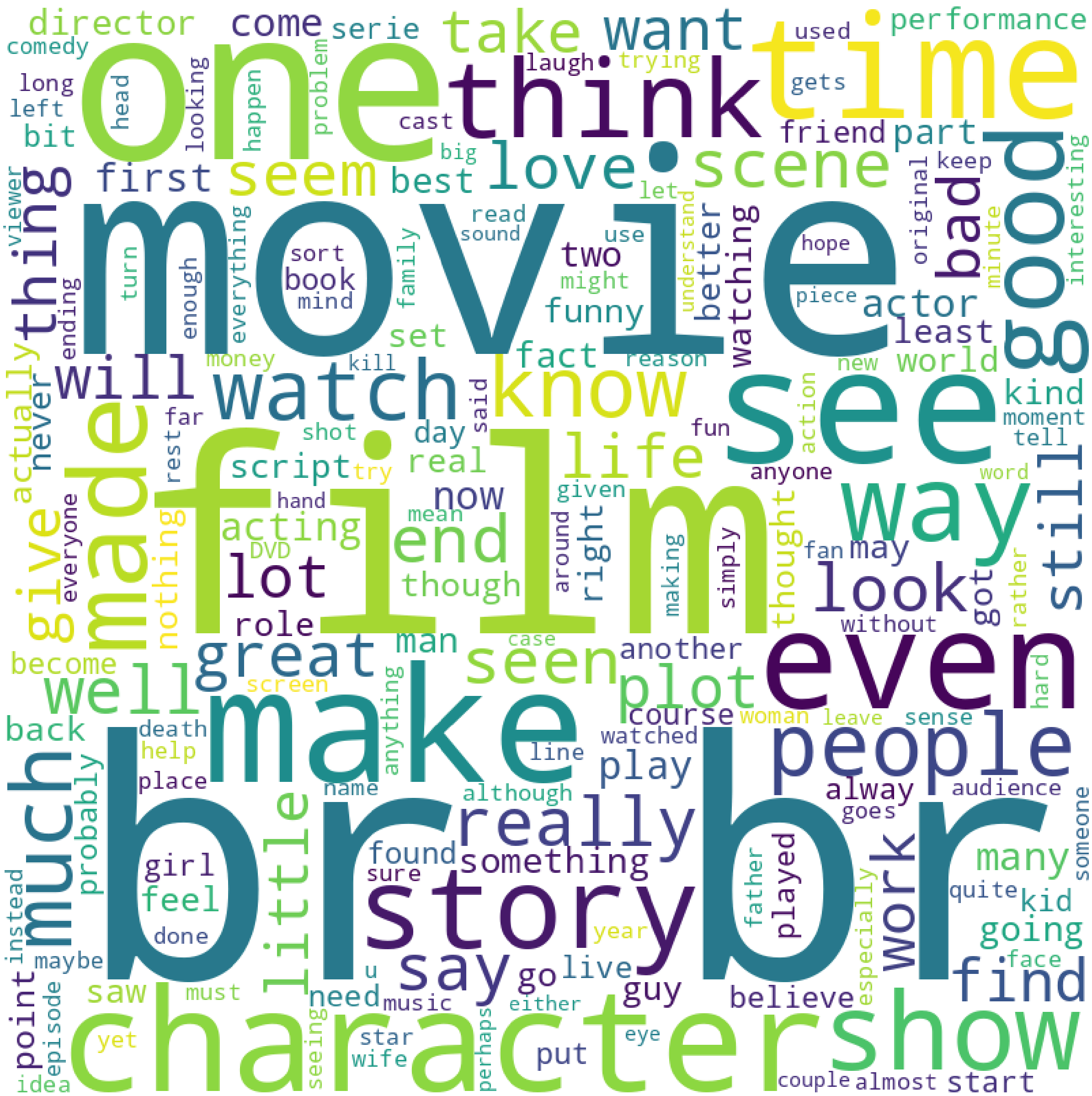
Out[ ]:

	0
review	object
sentiment	object

dtype: object

In [ ]:

```
# word cloud

from wordcloud import WordCloud
import matplotlib.pyplot as plt


# Concatenating all reviews into a single string
text = ' '.join(df['review'].tolist())

# Generating the word cloud
wordcloud = WordCloud(width=800, height=800, background_color='white').gener

# Plotting the word cloud
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



In [ ]:
```
# instll n import contractions library
!pip install contractions

from contractions import contractions_dict
```

```
Requirement already satisfied: contractions in /usr/local/lib/python3.11/dis
t-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3.
11/dist-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in /usr/local/lib/python3.11/dist-pa
ckages (from textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.11/di
st-packages (from textsearch>=0.0.21->contractions) (2.1.0)
```

In [ ]:
```
# Defining the stopword list and the lemmatizer
stopwords_list = stopwords.words('english')
lemmatizer = WordNetLemmatizer()
```

In [ ]:
```
# Defining a function for preprocessing
def preprocess_text(text):

    # Removing HTML tags
    text = re.sub('<[^>]+>', '', text)

    # Converting to lower case
    text = text.lower()

    # Expanding contractions
    words = []
    for word in word_tokenize(text):      #tokenization
        if word in contractions_dict:
            words.extend(word_tokenize(contractions_dict[word]))
        else:
            words.append(word)

    # Removing punctuation
    words = [word for word in words if word.isalpha()]

    # Removing stopwords
    words = [word for word in words if word not in stopwords_list]

    # Lemmatizing
    words = [lemmatizer.lemmatize(word) for word in words]

    # Joining the words back into a single string
    text = ' '.join(words)

    return text
```

In [ ]:
```
df['preprocessed_review'] = df['review'].apply(lambda x: preprocess_text(x))
#adding the new column for preprocessed words
```

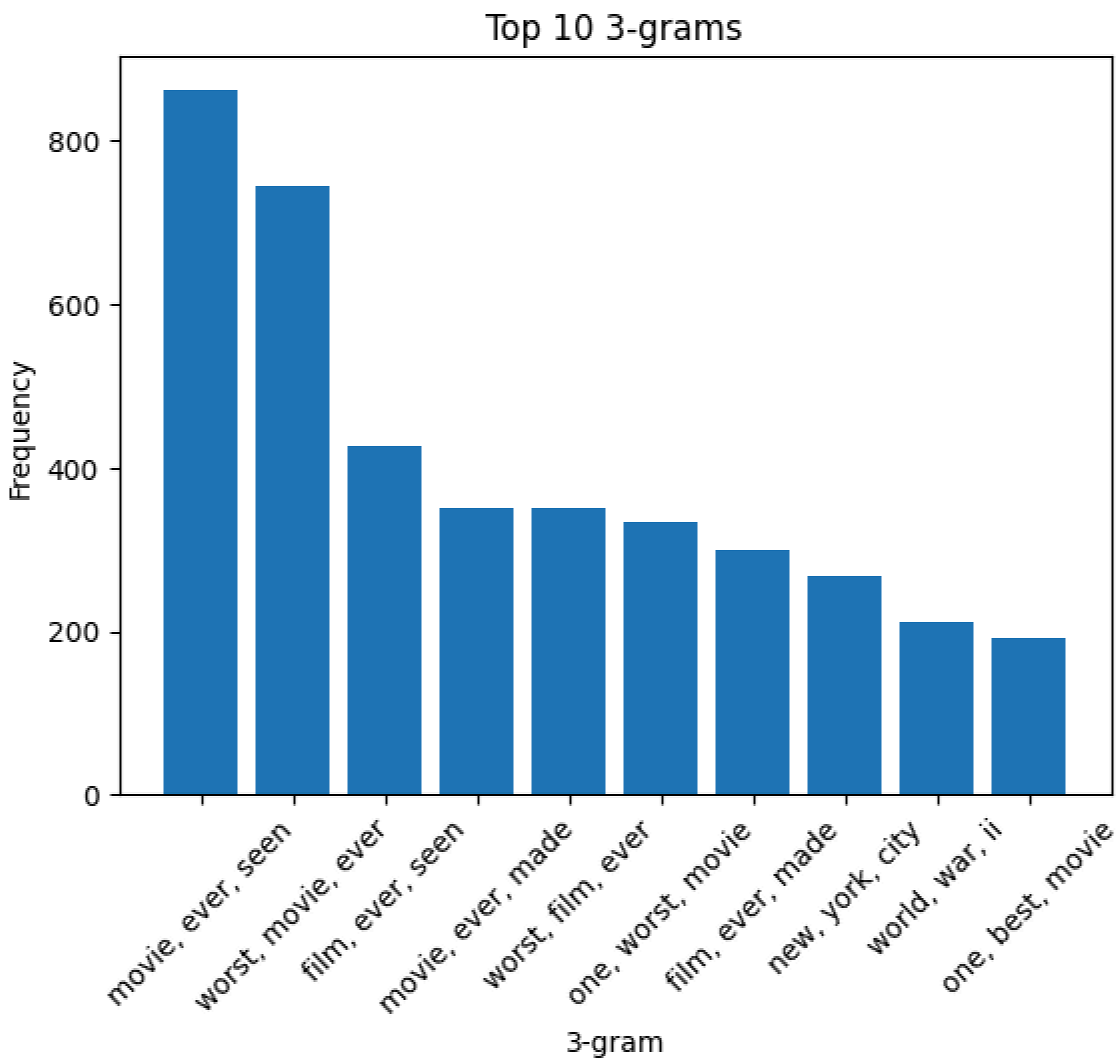
In [ ]:
```
df.head(4)
```

Out[ ]:

	review	sentiment	preprocessed_review
0	One of the other reviewers has mentioned that ...	positive	one reviewer mentioned watching oz episode hoo...
1	A wonderful little production. The...	positive	wonderful little production filming technique ...
2	I thought this was a wonderful way to spend ti...	positive	thought wonderful way spend time hot summer we...
3	Basically there's a family where a little boy ...	negative	basically family little boy jake think zombie ...

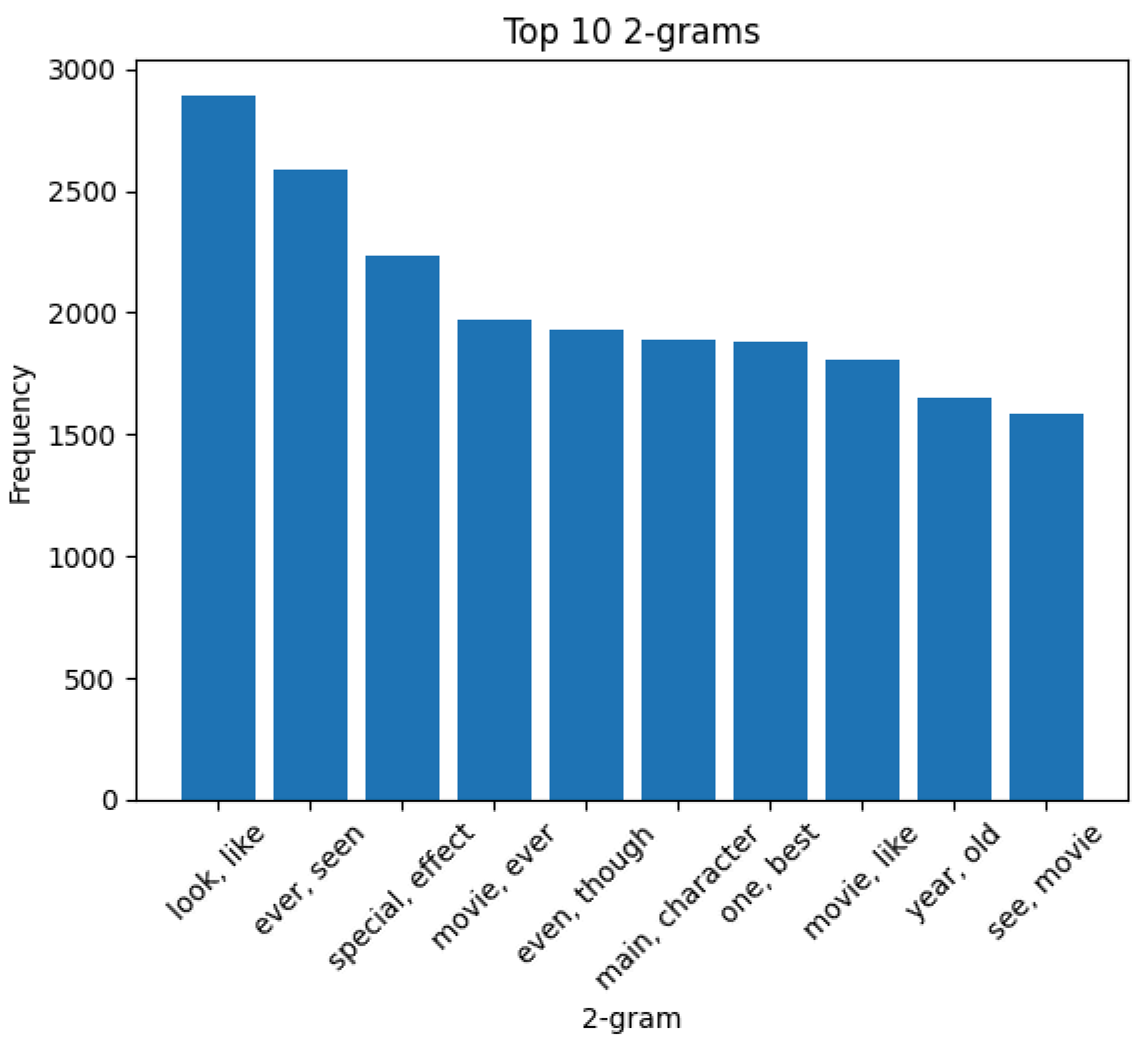
In [ ]:
```
from nltk import ngrams
from collections import Counter

# Define the text
```

```
text = " ".join(df.preprocessed_review.tolist())

# Generating the tri-grams
n = 3
ngrams_list = ngrams(text.split(), n)
ngrams_freq = Counter(ngrams_list)

# Plotting the top 10 most frequent words
top_ngrams = ngrams_freq.most_common(10)
x_labels = [', '.join(words) for words, count in top_ngrams]
y_values = [count for words, count in top_ngrams]

plt.bar(x_labels, y_values)
plt.xticks(rotation=45)
plt.xlabel(f'{n}-gram')
plt.ylabel('Frequency')
plt.title(f'Top 10 {n}-grams')
plt.show()
```



In [ ]:
```
# Define the text
text = " ".join(df.preprocessed_review.tolist())

# Generating the bi-grams
```

```
n = 2
ngrams_list = ngrams(text.split(), n)
ngrams_freq = Counter(ngrams_list)

# Plotting the top 10 most frequent words
top_ngrams = ngrams_freq.most_common(10)
x_labels = [', '.join(words) for words, count in top_ngrams]
y_values = [count for words, count in top_ngrams]

plt.bar(x_labels, y_values)
plt.xticks(rotation=45)
plt.xlabel(f'{n}-gram')
plt.ylabel('Frequency')
plt.title(f'Top 10 {n}-grams')
plt.show()
```



In [ ]:

```

```

b)

In [ ]:

```
#imports
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, LSTM, GRU, SimpleRNN,
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
```

In [ ]:
```
labels = np.array(df['sentiment'].map({'positive': 1, 'negative': 0})) # lab
```

In [ ]:
```
# splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['preprocessed_review'
```

In [ ]:
```
# Preprocess the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)
word_index = tokenizer.word_index
max_length= 100 # taking the max length as 100
train_data = pad_sequences(train_sequences, maxlen=max_length)
test_data = pad_sequences(test_sequences, maxlen=max_length)
```

In [ ]:
```
train_data.shape, test_data.shape
```

Out[ ]:
```
((40000, 100), (10000, 100))
```

grugrrnnrrrrvd

In [ ]:
```
X_train_tokens = [text.split() for text in X_train]

w2v_rnn = Word2Vec(sentences=X_train_tokens,
                   vector_size=100,
                   window=5,
                   min_count=1,
                   workers=4,
                   sg=0)  # CBOW

embedding_dim = 100
embedding_matrix_rnn = np.zeros((len(word_index) + 1, embedding_dim))

for word, i in word_index.items():
    if word in w2v_rnn.wv:
        embedding_matrix_rnn[i] = w2v_rnn.wv[word]
```

In [ ]:
```
#build and compile the RNN model
model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=len(word_index) + 1,
                        output_dim=embedding_dim,
                        weights=[embedding_matrix_rnn],
                        input_length=max_length,
                        trainable=False))
model_rnn.add(SimpleRNN(128))
```

```
model_rnn.add(Dense(1, activation='sigmoid'))

model_rnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
```

In [ ]:
```
#train the RNN model
history_rnn = model_rnn.fit(train_data, y_train,
                            epochs=100,
                            batch_size=64,
                            validation_split=0.2,
                            callbacks=[EarlyStopping(monitor='val_loss', pat
```

```
Epoch 1/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - accuracy: 0.6877 - loss: 0.5976
- val_accuracy: 0.6008 - val_loss: 0.7298
Epoch 2/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7202 - loss: 0.5598
- val_accuracy: 0.7551 - val_loss: 0.5185
Epoch 3/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7175 - loss: 0.5588
- val_accuracy: 0.6974 - val_loss: 0.6036
Epoch 4/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6798 - loss: 0.5941
- val_accuracy: 0.7046 - val_loss: 0.5758
Epoch 5/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7191 - loss: 0.5581
- val_accuracy: 0.7120 - val_loss: 0.5499
Epoch 6/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.5996 - loss: 0.7025
- val_accuracy: 0.7210 - val_loss: 0.5545
Epoch 7/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6754 - loss: 0.6019
- val_accuracy: 0.6856 - val_loss: 0.5977
Epoch 8/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6478 - loss: 0.6232
- val_accuracy: 0.7021 - val_loss: 0.5757
Epoch 9/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6914 - loss: 0.5879
- val_accuracy: 0.6992 - val_loss: 0.5756
Epoch 10/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7195 - loss: 0.5580
- val_accuracy: 0.6837 - val_loss: 0.5943
```

In [ ]:
```
#RNN model evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

y_pred_rnn = model_rnn.predict(test_data)
y_pred_rnn_labels = (y_pred_rnn > 0.5).astype(int)

rnn_acc = accuracy_score(y_test, y_pred_rnn_labels)
rnn_prec = precision_score(y_test, y_pred_rnn_labels)
rnn_rec = recall_score(y_test, y_pred_rnn_labels)
rnn_f1 = f1_score(y_test, y_pred_rnn_labels)

print("RNN Model Evaluation:")
print(f"Accuracy:  {rnn_acc:.4f}")
print(f"Precision: {rnn_prec:.4f}")
```

```
print(f"Recall:    {rnn_rec:.4f}")
print(f"F1 Score:  {rnn_f1:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
RNN Model Evaluation:
Accuracy:  0.6808
Precision: 0.7134
Recall:    0.6044
F1 Score:  0.6544
```

In [ ]:
```
#RNN model summary

model_rnn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Par
embedding (Embedding)	(64, 100, 100)	8,072
simple_rnn (SimpleRNN)	(64, 128)	29
dense (Dense)	(64, 1)	

Total params: 8,160,825 (31.13 MB)

Trainable params: 29,441 (115.00 KB)

Non-trainable params: 8,072,500 (30.79 MB)

Optimizer params: 58,884 (230.02 KB)

In [ ]:

LSTM

In [ ]:
```
# Build and compile the LSTM model
model_lstm = Sequential()
model_lstm.add(Embedding(input_dim=len(word_index) + 1,
                         output_dim=embedding_dim,
                         weights=[embedding_matrix_rnn],
                         input_length=max_length,
                         trainable=False))
model_lstm.add(LSTM(128, dropout=0.2))
model_lstm.add(Dense(1, activation='sigmoid'))

model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['a
```

In [ ]:
```
#train the LSTM Model
early_stop_lstm = EarlyStopping(monitor='val_loss', patience=8, restore_best

history_lstm = model_lstm.fit(train_data, y_train,
                              epochs=100,
                              batch_size=64,
                              validation_split=0.2,
```

```
                              callbacks=[early_stop_lstm],
                              verbose=1)
```

```
Epoch 1/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.7779 - loss: 0.4654 -
val_accuracy: 0.8334 - val_loss: 0.3653
Epoch 2/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8520 - loss: 0.3442 -
val_accuracy: 0.8558 - val_loss: 0.3390
Epoch 3/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8659 - loss: 0.3089 -
val_accuracy: 0.8591 - val_loss: 0.3260
Epoch 4/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8738 - loss: 0.2953 -
val_accuracy: 0.8680 - val_loss: 0.3065
Epoch 5/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8835 - loss: 0.2774 -
val_accuracy: 0.8708 - val_loss: 0.3120
Epoch 6/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8927 - loss: 0.2568 -
val_accuracy: 0.8666 - val_loss: 0.3135
Epoch 7/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8999 - loss: 0.2394 -
val_accuracy: 0.8661 - val_loss: 0.3095
Epoch 8/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9071 - loss: 0.2278 -
val_accuracy: 0.8683 - val_loss: 0.3192
Epoch 9/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9145 - loss: 0.2118 -
val_accuracy: 0.8644 - val_loss: 0.3348
Epoch 10/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9200 - loss: 0.1958 -
val_accuracy: 0.8658 - val_loss: 0.3230
Epoch 11/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9258 - loss: 0.1841 -
val_accuracy: 0.8665 - val_loss: 0.3457
Epoch 12/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9311 - loss: 0.1727 -
val_accuracy: 0.8683 - val_loss: 0.3501
```

In [ ]:

```
# LSTM Model Evaluation
y_pred_lstm = model_lstm.predict(test_data)
y_pred_lstm_labels = (y_pred_lstm > 0.5).astype(int)

lstm_acc = accuracy_score(y_test, y_pred_lstm_labels)
lstm_prec = precision_score(y_test, y_pred_lstm_labels)
lstm_rec = recall_score(y_test, y_pred_lstm_labels)
lstm_f1 = f1_score(y_test, y_pred_lstm_labels)

print("LSTM Model Evaluation:")
print(f"Accuracy:  {lstm_acc:.4f}")
print(f"Precision: {lstm_prec:.4f}")
print(f"Recall:    {lstm_rec:.4f}")
print(f"F1 Score:  {lstm_f1:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
LSTM Model Evaluation:
Accuracy:  0.8679
Precision: 0.8526
Recall:    0.8896
F1 Score:  0.8707
```

In [ ]:
```python
# LSTM Model summary
model_lstm.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Par
embedding_1 (Embedding)	(64, 100, 100)	8,072
lstm (LSTM)	(64, 128)	117
dense_1 (Dense)	(64, 1)	

Total params: 8,424,633 (32.14 MB)

Trainable params: 117,377 (458.50 KB)

Non-trainable params: 8,072,500 (30.79 MB)

Optimizer params: 234,756 (917.02 KB)

In [ ]:

GRU

In [ ]:
```python
# GRU model defining and compilation
model_gru = Sequential()
model_gru.add(Embedding(input_dim=len(word_index) + 1,
                        output_dim=embedding_dim,
                        weights=[embedding_matrix_rnn],
                        input_length=max_length,
                        trainable=False))

model_gru.add(GRU(128, dropout=0.2))
model_gru.add(Dense(1, activation='sigmoid'))

model_gru.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
```

In [ ]:
```python
#GRU model training

early_stop_gru = EarlyStopping(monitor='val_loss', patience=8, restore_best_

history_gru = model_gru.fit(train_data, y_train,
                            epochs=100,
                            batch_size=64,
                            validation_split=0.2,
                            callbacks=[early_stop_gru],
                            verbose=1)
```

```
Epoch 1/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.7547 - loss: 0.4847 -
val_accuracy: 0.8487 - val_loss: 0.3428
Epoch 2/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8636 - loss: 0.3259 -
val_accuracy: 0.8648 - val_loss: 0.3176
Epoch 3/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8685 - loss: 0.3047 -
val_accuracy: 0.8686 - val_loss: 0.3080
Epoch 4/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8764 - loss: 0.2923 -
val_accuracy: 0.8602 - val_loss: 0.3183
Epoch 5/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8854 - loss: 0.2766 -
val_accuracy: 0.8595 - val_loss: 0.3180
Epoch 6/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8905 - loss: 0.2632 -
val_accuracy: 0.8679 - val_loss: 0.3055
Epoch 7/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8994 - loss: 0.2442 -
val_accuracy: 0.8637 - val_loss: 0.3141
Epoch 8/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8995 - loss: 0.2379 -
val_accuracy: 0.8669 - val_loss: 0.3098
Epoch 9/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9157 - loss: 0.2126 -
val_accuracy: 0.8621 - val_loss: 0.3271
Epoch 10/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9200 - loss: 0.1971 -
val_accuracy: 0.8645 - val_loss: 0.3195
Epoch 11/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9270 - loss: 0.1821 -
val_accuracy: 0.8622 - val_loss: 0.3292
Epoch 12/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9277 - loss: 0.1774 -
val_accuracy: 0.8660 - val_loss: 0.3466
Epoch 13/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9370 - loss: 0.1574 -
val_accuracy: 0.8643 - val_loss: 0.3642
Epoch 14/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9402 - loss: 0.1520 -
val_accuracy: 0.8655 - val_loss: 0.3737
```

In [ ]:

```
#GRU model evaluation
y_pred_gru = model_gru.predict(test_data)
y_pred_gru_labels = (y_pred_gru > 0.5).astype(int)

gru_acc = accuracy_score(y_test, y_pred_gru_labels)
gru_prec = precision_score(y_test, y_pred_gru_labels)
gru_rec = recall_score(y_test, y_pred_gru_labels)
gru_f1 = f1_score(y_test, y_pred_gru_labels)

print("\nGRU Model Evaluation:")
print(f"Accuracy:  {gru_acc:.4f}")
print(f"Precision: {gru_prec:.4f}")
```

```
print(f"Recall:    {gru_rec:.4f}")
print(f"F1 Score:  {gru_f1:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

GRU Model Evaluation:
Accuracy:  0.8671
Precision: 0.8888
Recall:    0.8392
F1 Score:  0.8633
```

In [ ]:
```
#GRU model summary


model_gru.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Par
embedding_2 (Embedding)	(64, 100, 100)	8,072
gru (GRU)	(64, 128)	88
dense_2 (Dense)	(64, 1)	

Total params: 8,337,849 (31.81 MB)

Trainable params: 88,449 (345.50 KB)

Non-trainable params: 8,072,500 (30.79 MB)

Optimizer params: 176,900 (691.02 KB)

In [ ]:

BiLSTM

In [ ]:
```
# Defining and compiling th  BiLSTM model
model_bilstm = Sequential()
model_bilstm.add(Embedding(input_dim=len(word_index) + 1,
                           output_dim=embedding_dim,
                           weights=[embedding_matrix_rnn],
                           input_length=max_length,
                           trainable=False))

model_bilstm.add(Bidirectional(LSTM(128, dropout=0.2)))
model_bilstm.add(Dense(1, activation='sigmoid'))

model_bilstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
```

In [ ]:
```
# BiLSTM model training
early_stop_bilstm = EarlyStopping(monitor='val_loss', patience=6, restore_be

history_bilstm = model_bilstm.fit(train_data, y_train,
                                  epochs=100,
                                  batch_size=64,
```

```
                                    validation_split=0.2,
                                    callbacks=[early_stop_bilstm],
                                    verbose=1)
```

```
Epoch 1/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 8s 12ms/step - accuracy: 0.7829 - loss: 0.4536
- val_accuracy: 0.8366 - val_loss: 0.3630
Epoch 2/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.8495 - loss: 0.3456
- val_accuracy: 0.8464 - val_loss: 0.3396
Epoch 3/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.8660 - loss: 0.3140
- val_accuracy: 0.8539 - val_loss: 0.3305
Epoch 4/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.8756 - loss: 0.2947
- val_accuracy: 0.8645 - val_loss: 0.3233
Epoch 5/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.8838 - loss: 0.2735
- val_accuracy: 0.8680 - val_loss: 0.3100
Epoch 6/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.8942 - loss: 0.2558
- val_accuracy: 0.8649 - val_loss: 0.3074
Epoch 7/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9025 - loss: 0.2372
- val_accuracy: 0.8687 - val_loss: 0.3199
Epoch 8/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9109 - loss: 0.2162
- val_accuracy: 0.8691 - val_loss: 0.3207
Epoch 9/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9203 - loss: 0.1956
- val_accuracy: 0.8741 - val_loss: 0.3151
Epoch 10/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9303 - loss: 0.1747
- val_accuracy: 0.8685 - val_loss: 0.3460
Epoch 11/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9350 - loss: 0.1656
- val_accuracy: 0.8639 - val_loss: 0.3399
Epoch 12/100
500/500 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9394 - loss: 0.1536
- val_accuracy: 0.8620 - val_loss: 0.3641
```

In [ ]:

```
#BiLSTM model evaluation
y_pred_bilstm = model_bilstm.predict(test_data)
y_pred_bilstm_labels = (y_pred_bilstm > 0.5).astype(int)

bilstm_acc = accuracy_score(y_test, y_pred_bilstm_labels)
bilstm_prec = precision_score(y_test, y_pred_bilstm_labels)
bilstm_rec = recall_score(y_test, y_pred_bilstm_labels)
bilstm_f1 = f1_score(y_test, y_pred_bilstm_labels)

print("\nBiLSTM Model Evaluation:")
print(f"Accuracy:  {bilstm_acc:.4f}")
print(f"Precision: {bilstm_prec:.4f}")
print(f"Recall:    {bilstm_rec:.4f}")
print(f"F1 Score:  {bilstm_f1:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step

BiLSTM Model Evaluation:
Accuracy:  0.8695
Precision: 0.8684
Recall:    0.8710
F1 Score:  0.8697
```

In [ ]:
```python
# BiLSTM model summary
model_bilstm.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Par
embedding_3 (Embedding)	(64, 100, 100)	8,072
bidirectional (Bidirectional)	(64, 256)	234
dense_3 (Dense)	(64, 1)	

Total params: 8,776,761 (33.48 MB)

Trainable params: 234,753 (917.00 KB)

Non-trainable params: 8,072,500 (30.79 MB)

Optimizer params: 469,508 (1.79 MB)

In [ ]:

In [ ]:
```python
import pandas as pd
from IPython.display import Markdown
from IPython.display import display

df_all_results = pd.DataFrame([
    ["RNN", rnn_acc, rnn_prec, rnn_rec, rnn_f1],
    ["LSTM", lstm_acc, lstm_prec, lstm_rec, lstm_f1],
    ["GRU", gru_acc, gru_prec, gru_rec, gru_f1],
    ["BiLSTM", bilstm_acc, bilstm_prec, bilstm_rec, bilstm_f1]
], columns=["Model", "Accuracy", "Precision", "Recall", "F1 Score"]).round(4
display(Markdown("### Final Results Table"))
display(df_all_results)
```

Final Results Table

	Model	Accuracy	Precision	Recall	F1 Score
0	RNN	0.6808	0.7134	0.6044	0.6544
1	LSTM	0.8679	0.8526	0.8896	0.8707
2	GRU	0.8671	0.8888	0.8392	0.8633
3	BiLSTM	0.8695	0.8684	0.8710	0.8697

- From the above results we can say that models like LSTM, GRU, and BiLSTM outperformed the basic RNN, especially in recall and F1 score.
- BiLSTM performed best overall due to its ability to capture context in both directions. GRU showed strong precision, making fewer false positives.
- Using Gensim-trained Word2Vec embeddings helped all models start with meaningful word representations, improving performance across the board.

In [ ]:

Hyperparameters used:

Hyperparameter	RNN	LSTM	GRU	BiLSTM
Hidden Layer Type	SimpleRNN(128)	LSTM(128, dropout=0.2)	GRU(128, dropout=0.2)	Bidirectional(LSTM(128, dropout=0.2))
Activation (Hidden)	tanh (default in SimpleRNN)	tanh (default in LSTM)	tanh (default in GRU)	tanh (default in LSTM)
Activation (Output)	sigmoid	sigmoid	sigmoid	sigmoid
Weight Initializer	glorot_uniform (default)	glorot_uniform (default)	glorot_uniform (default)	glorot_uniform (default)
Embedding Layer	Word2Vec, non-trainable	Word2Vec, non-trainable	Word2Vec, non-trainable	Word2Vec, non-trainable
Hidden Layers	1 RNN	1 LSTM	1 GRU	1 BiLSTM
Neurons in Hidden	128	128	128	128
Loss Function	binary_crossentropy	binary_crossentropy	binary_crossentropy	binary_crossentropy
Optimizer	Adam	Adam	Adam	Adam
Learning Rate	0.001 (default of Adam)	0.001 (default)	0.001 (default)	0.001 (default)
Epochs	100 (with early stopping)	100 (with early stopping)	100 (with early stopping)	100 (with early stopping)
Batch Size	64	64	64	64
Early Stopping Patience	8	8	8	6
Validation Split	0.2	0.2	0.2	0.2
Evaluation Metrics	Accuracy (via Keras) & Precision, Recall, F1 Score (via sklearn on test set)	Accuracy (via Keras) & Precision, Recall, F1 Score (via sklearn on test set)	Accuracy (via Keras) & Precision, Recall, F1 Score (via sklearn on test set)	Accuracy (via Keras) & Precision, Recall, F1 Score (via sklearn on test set)

In [ ]:

Problem 2) DCGAN on PathMnist dataset

In [2]:
```
import torch

# Set device to GPU if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

In [3]:
```
!pip install medmnist tqdm
```

```
Requirement already satisfied: medmnist in /usr/local/lib/python3.11/dist-pa
ckages (3.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packag
es (4.67.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packa
ges (from medmnist) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-pack
ages (from medmnist) (2.2.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dis
t-packages (from medmnist) (1.2.2)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dis
t-packages (from medmnist) (0.25.1)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-pack
ages (from medmnist) (11.1.0)
Requirement already satisfied: fire in /usr/local/lib/python3.11/dist-packag
es (from medmnist) (0.7.0)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packa
ges (from medmnist) (2.5.1+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist
-packages (from medmnist) (0.20.1+cu124)
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-p
ackages (from fire->medmnist) (2.5.0)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.11/dist-pac
kages (from numpy->medmnist) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.11/dist-
packages (from numpy->medmnist) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.11/dist-p
ackages (from numpy->medmnist) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-package
s (from numpy->medmnist) (2025.1.0)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/dist-pack
ages (from numpy->medmnist) (2022.1.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.11/dist
-packages (from numpy->medmnist) (2.4.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyth
on3.11/dist-packages (from pandas->medmnist) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dis
t-packages (from pandas->medmnist) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/d
ist-packages (from pandas->medmnist) (2025.2)
Requirement already satisfied: scipy>=1.11.2 in /usr/local/lib/python3.11/di
st-packages (from scikit-image->medmnist) (1.15.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/di
st-packages (from scikit-image->medmnist) (3.4.2)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/pyth
on3.11/dist-packages (from scikit-image->medmnist) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python
3.11/dist-packages (from scikit-image->medmnist) (2025.1.10)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/di
st-packages (from scikit-image->medmnist) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.1
1/dist-packages (from scikit-image->medmnist) (0.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/di
st-packages (from scikit-learn->medmnist) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python
3.11/dist-packages (from scikit-learn->medmnist) (3.6.0)
```

```
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
ckages (from torch->medmnist) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/py
thon3.11/dist-packages (from torch->medmnist) (4.13.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pack
ages (from torch->medmnist) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-pack
ages (from torch->medmnist) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch->medmnist) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/lo
cal/lib/python3.11/dist-packages (from torch->medmnist) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch->medmnist) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/li
b/python3.11/dist-packages (from torch->medmnist) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/li
b/python3.11/dist-packages (from torch->medmnist) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/li
b/python3.11/dist-packages (from torch->medmnist) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/
lib/python3.11/dist-packages (from torch->medmnist) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/
lib/python3.11/dist-packages (from torch->medmnist) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/loca
l/lib/python3.11/dist-packages (from torch->medmnist) (12.3.1.170)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/py
thon3.11/dist-packages (from torch->medmnist) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch->medmnist) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch->medmnist) (12.4.127)
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/di
st-packages (from torch->medmnist) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/di
st-packages (from torch->medmnist) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
11/dist-packages (from sympy==1.13.1->torch->medmnist) (1.3.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pa
ckages (from python-dateutil>=2.8.2->pandas->medmnist) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/
dist-packages (from jinja2->torch->medmnist) (3.0.2)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/local/lib/py
thon3.11/dist-packages (from mkl->numpy->medmnist) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.11/dist
-packages (from mkl->numpy->medmnist) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.11/dist
-packages (from tbb==2022.*->mkl->numpy->medmnist) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.
11/dist-packages (from mkl_umath->numpy->medmnist) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/li
b/python3.11/dist-packages (from intel-openmp<2026,>=2024->mkl->numpy->medmn
ist) (2024.2.0)
```

In [4]:

```
#imports
import os
```

```python
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision.utils import save_image, make_grid
from tqdm import tqdm
import medmnist
from medmnist import PathMNIST
from scipy import linalg
```

In [5]:

```python
# random seed
def set_random_seed(seed_value=42):
    torch.manual_seed(seed_value)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed_value)
    np.random.seed(seed_value)
    return seed_value

seed = set_random_seed(42)
```

In [6]:

```python
# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

if device.type == "cuda":
    cuda_device_name = torch.cuda.get_device_name(0)
    cuda_memory = torch.cuda.get_device_properties(0).total_memory / 1e9
    print(f"CUDA Device: {cuda_device_name}")
    print(f"CUDA Memory: {cuda_memory:.2f} GB")
```

```
Using device: cuda
CUDA Device: Tesla P100-PCIE-16GB
CUDA Memory: 17.06 GB
```

In [7]:
```python
# Hyperparameters
batch_size = 64
lr = 0.0002
beta1 = 0.5
beta2 = 0.999
z_dim = 100
image_size = 28  # PathMNIST is 28x28
channels = 3  # PathMNIST has 3 color channels
epochs = 1000  # 1000 epochs

print("Hyperparameters:")
print(f"Batch size: {batch_size}")
print(f"Learning rate: {lr}")
print(f"Beta1: {beta1}, Beta2: {beta2}")
print(f"Latent dimension: {z_dim}")
print(f"Image size: {image_size}")
print(f"Channels: {channels}")
print(f"Epochs: {epochs}")
```

```
Hyperparameters:
Batch size: 64
Learning rate: 0.0002
Beta1: 0.5, Beta2: 0.999
Latent dimension: 100
Image size: 28
Channels: 3
Epochs: 1000
```

In [8]:
```python
def load_dataset():
    print("Loading MedMNIST dataset...")

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    try:
        # loading the PathMNIST dataset
        print("Loading training dataset...")
        train_dataset = PathMNIST(split="train", transform=transform, downlo
        print("Training dataset loaded successfully.")

        print("Loading test dataset...")
        test_dataset = PathMNIST(split="test", transform=transform, downloac
        print("Test dataset loaded successfully.")

        print(f"Dataset size: {len(train_dataset)}")
        print(f"Image shape: {train_dataset[0][0].shape}")

        sample_img = train_dataset[0][0]
        print(f"Sample image range: Min={sample_img.min():.4f}, Max={sample_
        print("Creating data loaders...")
        train_loader = DataLoader(
            train_dataset, batch_size=batch_size, shuffle=True, num_workers=
        )
        test_loader = DataLoader(
```

```
            test_dataset, batch_size=batch_size, shuffle=False, num_workers=
        )
        print("Data loaders created successfully.")

        return train_dataset, test_dataset, train_loader, test_loader
    except Exception as e:
        print(f"Error loading dataset: {str(e)}")
        print(traceback.format_exc())
        raise

train_dataset, test_dataset, train_loader, test_loader = load_dataset()
```

```
Loading MedMNIST dataset...
Loading training dataset...
Using downloaded and verified file: /root/.medmnist/pathmnist.npz
Training dataset loaded successfully.
Loading test dataset...
Using downloaded and verified file: /root/.medmnist/pathmnist.npz
Test dataset loaded successfully.
Dataset size: 89996
Image shape: torch.Size([3, 28, 28])
Sample image range: Min=0.2863, Max=0.7882
Creating data loaders...
Data loaders created successfully.
```

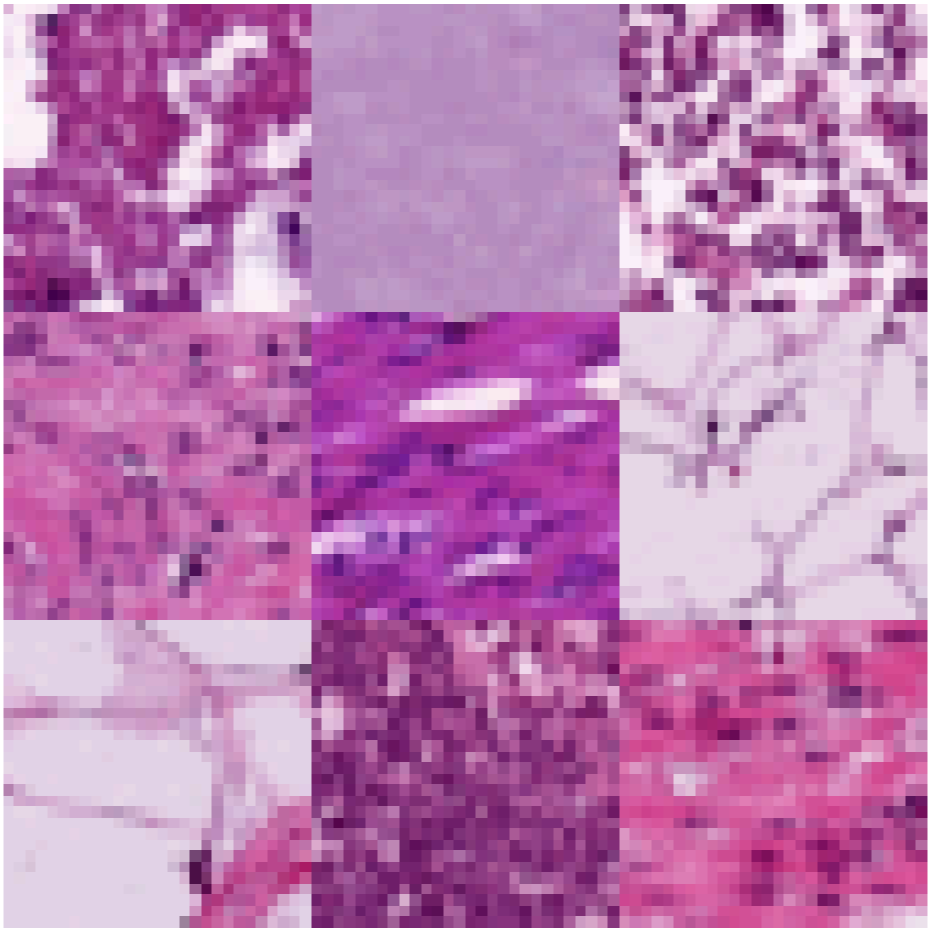
In [9]:

```
# checking the image dimension
sample_img, sample_label = train_dataset[0]
print(f"Sample image shape: {sample_img.shape}")
```

```
Sample image shape: torch.Size([3, 28, 28])
```

In [10]:

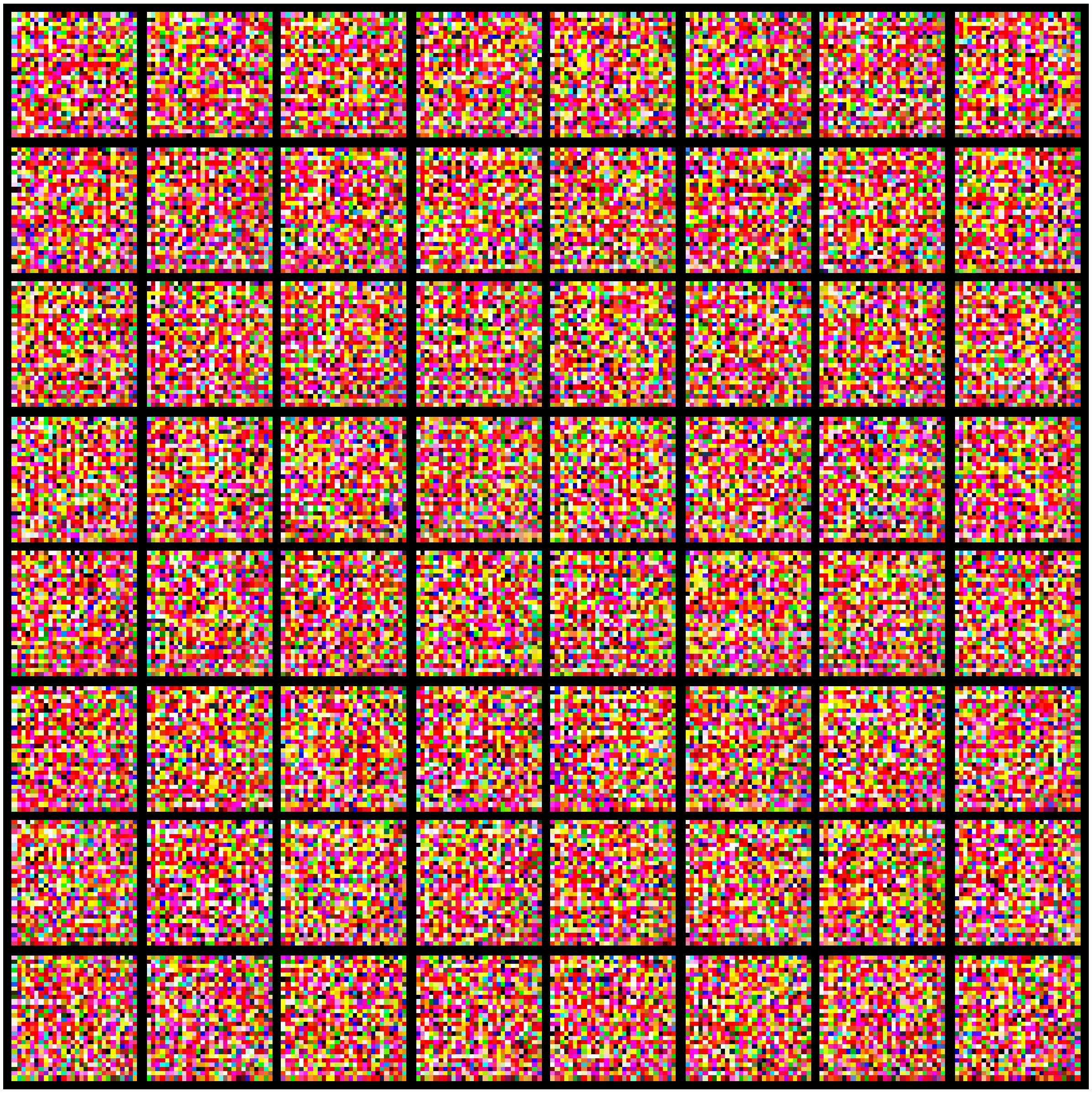
```
# displaying the first 9 images from the dataset
def show_first_9_images(train_loader):
    images, labels = next(iter(train_loader))
    grid = make_grid(images[:9], nrow=3, padding=0, normalize=True)
    plt.figure(figsize=(6, 6))
    plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
    plt.axis('off')
    plt.show()

show_first_9_images(train_loader)
```



In [11]:
```
import matplotlib.pyplot as plt
import torch
import torch.nn.functional as F
from torchvision.utils import make_grid

# print grid of images
def display_images(image_batch, nrow=8):
    """Displays a grid of images"""
    grid_img = make_grid(image_batch, nrow=nrow, normalize=True)
    plt.figure(figsize=(10, 10))
    plt.imshow(grid_img.permute(1, 2, 0).cpu().numpy())
    plt.axis('off')
    plt.show()

class Generator(nn.Module):
    def __init__(self, z_dim, channels=3):
        super(Generator, self).__init__()
        self.z_dim = z_dim

        self.project = nn.Linear(z_dim, 1024 * 4 * 4)

        # Transposed convolution blocks
        self.conv1 = nn.ConvTranspose2d(1024, 512, kernel_size=5, stride=2,
        self.bn1 = nn.BatchNorm2d(512)

        self.conv2 = nn.ConvTranspose2d(512, 256, kernel_size=5, stride=2, p
```

```
        self.bn2 = nn.BatchNorm2d(256)

        self.conv3 = nn.ConvTranspose2d(256, 128, kernel_size=5, stride=2, p
        self.bn3 = nn.BatchNorm2d(128)

        # final convolution layer match the real input images
        self.conv4 = nn.ConvTranspose2d(128, channels, kernel_size=5, stride

    def forward(self, z):
        x = self.project(z)

        x = x.view(-1, 1024, 4, 4)

        # Transposed convolution blocks with ReLU activation
        x = self.conv1(x)
        x = self.bn1(x)
        x = F.relu(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = F.relu(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = F.relu(x)

        # Final layer with tanh activation to get values in [-1, 1]
        x = self.conv4(x)
        x = torch.tanh(x)

        return x

# generator model
z_dim = 100
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
generator = Generator(z_dim).to(device)
print(device)  # Should say: cuda


batch_size = 64
latent_vectors = torch.randn(batch_size, z_dim).to(device)

generated_images = generator(latent_vectors)

# Display the generated images
display_images(generated_images)
print("Generated image batch shape:", generated_images.shape)
```

```
cuda
```



Generated image batch shape: torch.Size([64, 3, 28, 28])

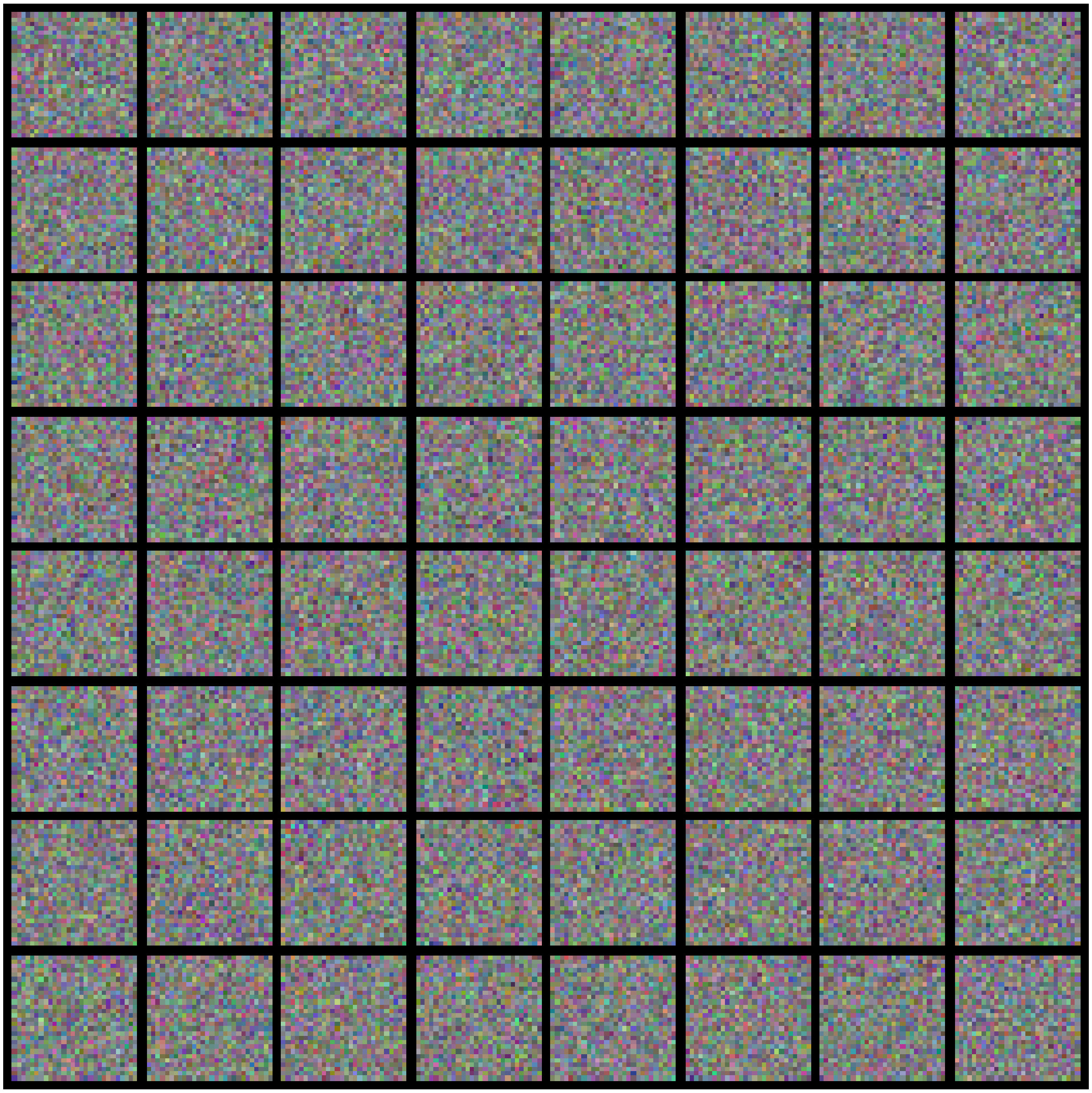
In [12]:
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from torchvision.utils import make_grid

class Discriminator(nn.Module):
    def __init__(self, channels=3):
        super(Discriminator, self).__init__()

        # 1st convo layer
        self.conv1 = nn.Conv2d(channels, 64, kernel_size=5, stride=2, paddir
        self.lrelu1 = nn.LeakyReLU(0.2, inplace=True)

        # 2nd convo layer
        self.conv2 = nn.Conv2d(64, 128, kernel_size=5, stride=2, padding=2)
        self.bn2 = nn.BatchNorm2d(128)
        self.lrelu2 = nn.LeakyReLU(0.2, inplace=True)
```

```
        # 3rd convo layer
        self.conv3 = nn.Conv2d(128, 256, kernel_size=5, stride=2, padding=2)
        self.bn3 = nn.BatchNorm2d(256)
        self.lrelu3 = nn.LeakyReLU(0.2, inplace=True)

        # 4th convo layer
        self.conv4 = nn.Conv2d(256, 512, kernel_size=5, stride=2, padding=2)
        self.bn4 = nn.BatchNorm2d(512)
        self.lrelu4 = nn.LeakyReLU(0.2, inplace=True)

        # 5th convo layer
        self.conv5 = nn.Conv2d(512, 1024, kernel_size=5, stride=2, padding=2
        self.bn5 = nn.BatchNorm2d(1024)
        self.lrelu5 = nn.LeakyReLU(0.2, inplace=True)

        # Output layer
        self.flatten = nn.Flatten()
        self.dense = None

    def forward(self, x):
        x = self.conv1(x)
        x = self.lrelu1(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.lrelu2(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = self.lrelu3(x)

        x = self.conv4(x)
        x = self.bn4(x)
        x = self.lrelu4(x)

        x = self.conv5(x)
        x = self.bn5(x)
        x = self.lrelu5(x)

        x = self.flatten(x)

        if self.dense is None:
            self.dense = nn.Linear(x.shape[1], 1).to(x.device)
            print(f"Initialized dense layer with input size: {x.shape[1]}")

        x = self.dense(x)

        return x

def test_discriminator():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    discriminator = Discriminator(channels=3).to(device)
    batch_size = 64
    test_images = torch.randn(batch_size, 3, 28, 28).to(device)
    output = discriminator(test_images)
    print("Discriminator output shape:", output.shape)
```

```python
    display_images(test_images)

    return discriminator

#  64 grid of images
def display_images(image_batch, nrow=8):
    """Displays a grid of images"""
    grid_img = make_grid(image_batch[:64], nrow=nrow, normalize=True)
    plt.figure(figsize=(10, 10))
    plt.imshow(grid_img.permute(1, 2, 0).cpu().numpy())
    plt.axis('off')
    plt.title("Sample Images")
    plt.show()

if __name__ == "__main__":
    test_discriminator()
```

```
Initialized dense layer with input size: 1024
Discriminator output shape: torch.Size([64, 1])
```

Sample Images



In [13]:
```
# used Binary Cross Entropy with logits (raw output)
criterion = nn.BCEWithLogitsLoss()

# Generator loss
def generator_loss(discriminator, fake_images):
    # fooling the discriminator — label as real (1)
    labels = torch.ones(fake_images.size(0), 1).to(fake_images.device)
    outputs = discriminator(fake_images)
    loss = criterion(outputs, labels)
    return loss

# Discriminator loss
def discriminator_loss(discriminator, real_images, fake_images):
    real_labels = torch.ones(real_images.size(0), 1).to(real_images.device)
    real_outputs = discriminator(real_images)
    real_loss = criterion(real_outputs, real_labels)

    fake_labels = torch.zeros(fake_images.size(0), 1).to(fake_images.device)
```

```python
    fake_outputs = discriminator(fake_images.detach())
    fake_loss = criterion(fake_outputs, fake_labels)

    return real_loss + fake_loss
```

In [14]:
```python
generator = Generator(z_dim).to(device)
discriminator = Discriminator(channels=3).to(device)
print(device)  # Should say: cuda


# Optimizers
lr = 0.0002
beta1 = 0.5
beta2 = 0.999

generator_optimizer = torch.optim.Adam(generator.parameters(), lr=lr, betas=
discriminator_optimizer = torch.optim.Adam(discriminator.parameters(), lr=lr
```

```
cuda
```

In [15]:
```python
def train_discriminator(generator, discriminator, real_images, z, optimizer)
    real_images = real_images.to(device)
    z = z.to(device)
    fake_images = generator(z)

    # Compute loss
    loss = discriminator_loss(discriminator, real_images, fake_images)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    return loss.item()
```

In [16]:
```python
# training generator step
def train_generator(generator, discriminator, z_batch, optimizer):
    generator.train()
    discriminator.eval()

    optimizer.zero_grad()

    fake_images = generator(z_batch)

    # Compute generator loss
    g_loss = generator_loss(discriminator, fake_images)
    g_loss.backward()
    optimizer.step()

    return g_loss.item()
```

In [17]:
```python
!pip install torchsummary

from torchsummary import summary

generator = Generator(z_dim).to(device)
discriminator = Discriminator(channels=3).to(device)
print(device)  # Should say: cuda
```

```
# Print model summaries
print("Generator Summary:")
summary(generator, input_size=(z_dim,))

print("\nDiscriminator Summary:")
summary(discriminator, input_size=(3, 28, 28))
```

```
Requirement already satisfied: torchsummary in /usr/local/lib/python3.11/dis
t-packages (1.5.1)
cuda
Generator Summary:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 16384]       1,654,784
   ConvTranspose2d-2            [-1, 512, 8, 8]      13,107,712
       BatchNorm2d-3            [-1, 512, 8, 8]           1,024
   ConvTranspose2d-4          [-1, 256, 16, 16]       3,277,056
       BatchNorm2d-5          [-1, 256, 16, 16]             512
   ConvTranspose2d-6          [-1, 128, 32, 32]         819,328
       BatchNorm2d-7          [-1, 128, 32, 32]             256
   ConvTranspose2d-8            [-1, 3, 28, 28]           9,603
================================================================
Total params: 18,870,275
Trainable params: 18,870,275
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 3.64
Params size (MB): 71.98
Estimated Total Size (MB): 75.63
----------------------------------------------------------------

Discriminator Summary:
Initialized dense layer with input size: 1024
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 14, 14]           4,864
         LeakyReLU-2           [-1, 64, 14, 14]               0
            Conv2d-3            [-1, 128, 7, 7]         204,928
       BatchNorm2d-4            [-1, 128, 7, 7]             256
         LeakyReLU-5            [-1, 128, 7, 7]               0
            Conv2d-6            [-1, 256, 4, 4]         819,456
       BatchNorm2d-7            [-1, 256, 4, 4]             512
         LeakyReLU-8            [-1, 256, 4, 4]               0
            Conv2d-9            [-1, 512, 2, 2]       3,277,312
      BatchNorm2d-10            [-1, 512, 2, 2]           1,024
        LeakyReLU-11            [-1, 512, 2, 2]               0
           Conv2d-12           [-1, 1024, 1, 1]      13,108,224
      BatchNorm2d-13           [-1, 1024, 1, 1]           2,048
        LeakyReLU-14           [-1, 1024, 1, 1]               0
          Flatten-15                 [-1, 1024]               0
================================================================
Total params: 17,418,624
Trainable params: 17,418,624
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.51
Params size (MB): 66.45
Estimated Total Size (MB): 66.96
----------------------------------------------------------------
```

In [18]:
```
from tqdm import tqdm

# used learning rate and early stopping with patience=20
lr = 0.0002
patience = 40
best_gen_loss = float('inf')
patience_counter = 0

print(device)  # Should say: cuda

generator_optimizer = torch.optim.Adam(generator.parameters(), lr=lr, betas=
discriminator_optimizer = torch.optim.Adam(discriminator.parameters(), lr=lr

# Track losses
gen_loss_profile = []
disc_loss_profile = []

fixed_noise = torch.randn(64, z_dim).to(device)

# Training loop
num_epochs = 1000

for epoch in range(num_epochs):
    g_running_loss = 0.0
    d_running_loss = 0.0

    loop = tqdm(train_loader, leave=False, desc=f"Epoch [{epoch+1}/{num_epoc

    for real_images, _ in loop:
        real_images = real_images.to(device)
        batch_size = real_images.size(0)
        z = torch.randn(batch_size, z_dim).to(device)

        # Train discriminator
        d_loss = train_discriminator(generator, discriminator, real_images,

        # Train generator
        z = torch.randn(batch_size, z_dim).to(device)
        g_loss = train_generator(generator, discriminator, z, generator_opti

        d_running_loss += d_loss
        g_running_loss += g_loss

        loop.set_postfix({
            "Gen Loss": f"{g_loss:.4f}",
            "Disc Loss": f"{d_loss:.4f}"
        })

    avg_d_loss = d_running_loss / len(train_loader)
    avg_g_loss = g_running_loss / len(train_loader)

    disc_loss_profile.append(avg_d_loss)
    gen_loss_profile.append(avg_g_loss)

    print(f"Epoch [{epoch+1}/{num_epochs}] - Gen Loss: {avg_g_loss:.4f}, Dis
```

```
    if (epoch + 1) % 100 == 0:
        with torch.no_grad():
            fake_samples = generator(fixed_noise).detach().cpu()
        display_images(fake_samples)

    if avg_g_loss < best_gen_loss:
        best_gen_loss = avg_g_loss
        patience_counter = 0
    else:
        patience_counter += 1

    if patience_counter >= patience:
        print(f"Early stopping triggered at epoch {epoch+1}")
        break
```

```
cuda

Epoch [1/1000] - Gen Loss: 2.3975, Disc Loss: 0.8876

Epoch [2/1000] - Gen Loss: 2.2729, Disc Loss: 0.8867

Epoch [4/1000] - Gen Loss: 1.8988, Disc Loss: 1.0225

Epoch [7/1000] - Gen Loss: 1.8633, Disc Loss: 0.9641

Epoch [8/1000] - Gen Loss: 1.7669, Disc Loss: 1.0114

Epoch [10/1000] - Gen Loss: 1.6304, Disc Loss: 1.0388

Epoch [11/1000] - Gen Loss: 1.5727, Disc Loss: 1.0596

Epoch [12/1000] - Gen Loss: 1.4081, Disc Loss: 1.0996

Epoch [13/1000] - Gen Loss: 1.4050, Disc Loss: 1.0967

Epoch [14/1000] - Gen Loss: 1.3812, Disc Loss: 1.1053

Epoch [15/1000] - Gen Loss: 1.4371, Disc Loss: 1.1014

Epoch [16/1000] - Gen Loss: 1.2658, Disc Loss: 1.1539

Epoch [17/1000] - Gen Loss: 1.2767, Disc Loss: 1.1347

Epoch [18/1000] - Gen Loss: 1.3427, Disc Loss: 1.0975

Epoch [19/1000] - Gen Loss: 1.3710, Disc Loss: 1.0802

Epoch [20/1000] - Gen Loss: 1.4395, Disc Loss: 1.0642

Epoch [21/1000] - Gen Loss: 1.4673, Disc Loss: 1.0440
```

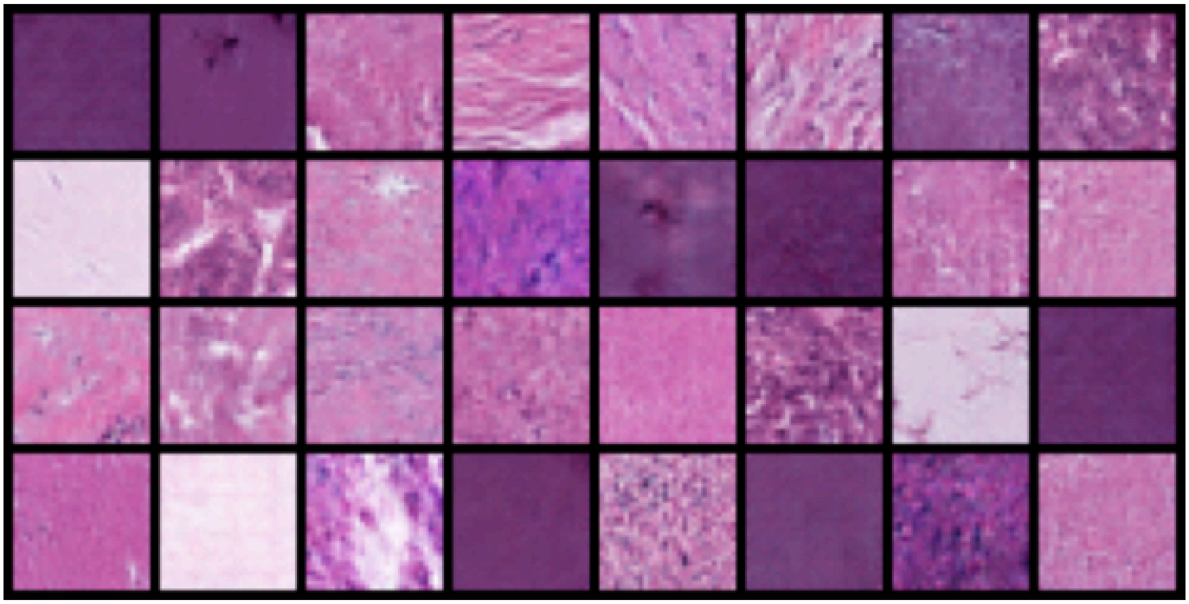
```
Epoch [22/1000] - Gen Loss: 1.4748, Disc Loss: 1.0324
Epoch [23/1000] - Gen Loss: 1.4509, Disc Loss: 1.0356
Epoch [24/1000] - Gen Loss: 1.4133, Disc Loss: 1.0295
Epoch [25/1000] - Gen Loss: 1.4068, Disc Loss: 1.0293
Epoch [26/1000] - Gen Loss: 1.3919, Disc Loss: 1.0256
Epoch [27/1000] - Gen Loss: 1.3992, Disc Loss: 1.0145
Epoch [28/1000] - Gen Loss: 1.4263, Disc Loss: 1.0004
Epoch [29/1000] - Gen Loss: 1.4239, Disc Loss: 0.9914
Epoch [30/1000] - Gen Loss: 1.4642, Disc Loss: 0.9790
Epoch [31/1000] - Gen Loss: 1.4864, Disc Loss: 0.9685
Epoch [32/1000] - Gen Loss: 1.5396, Disc Loss: 0.9474
Epoch [33/1000] - Gen Loss: 1.5633, Disc Loss: 0.9331
Epoch [34/1000] - Gen Loss: 1.6077, Disc Loss: 0.9127
Epoch [35/1000] - Gen Loss: 1.6837, Disc Loss: 0.8845
Epoch [36/1000] - Gen Loss: 1.7522, Disc Loss: 0.8525
Epoch [37/1000] - Gen Loss: 1.8152, Disc Loss: 0.8292
Epoch [38/1000] - Gen Loss: 1.8815, Disc Loss: 0.8110
Epoch [39/1000] - Gen Loss: 1.9646, Disc Loss: 0.7787
Epoch [40/1000] - Gen Loss: 2.0171, Disc Loss: 0.7610
Epoch [41/1000] - Gen Loss: 2.0907, Disc Loss: 0.7397
Epoch [42/1000] - Gen Loss: 2.1591, Disc Loss: 0.7190
Epoch [43/1000] - Gen Loss: 2.2495, Disc Loss: 0.6852
Epoch [44/1000] - Gen Loss: 2.3197, Disc Loss: 0.6665
Epoch [45/1000] - Gen Loss: 2.3826, Disc Loss: 0.6576
```

```
Epoch [46/1000] - Gen Loss: 2.4520, Disc Loss: 0.6329

Epoch [47/1000] - Gen Loss: 2.4980, Disc Loss: 0.6141

Epoch [48/1000] - Gen Loss: 2.5418, Disc Loss: 0.6035

Epoch [49/1000] - Gen Loss: 2.6116, Disc Loss: 0.5838

Epoch [50/1000] - Gen Loss: 2.6645, Disc Loss: 0.5727

Epoch [51/1000] - Gen Loss: 2.6971, Disc Loss: 0.5687

Epoch [52/1000] - Gen Loss: 2.7181, Disc Loss: 0.5605

Epoch [53/1000] - Gen Loss: 2.7292, Disc Loss: 0.5523

Epoch [54/1000] - Gen Loss: 2.7404, Disc Loss: 0.5481

Epoch [55/1000] - Gen Loss: 2.7565, Disc Loss: 0.5409

Epoch [56/1000] - Gen Loss: 2.7532, Disc Loss: 0.5329
Early stopping triggered at epoch 56
```

In [19]:

```
import matplotlib.pyplot as plt
from torchvision.utils import make_grid

def display_images(images, nrow=8):
    grid_img = make_grid(images[:32], nrow=nrow, normalize=True)
    plt.figure(figsize=(8, 8))
    plt.imshow(grid_img.permute(1, 2, 0).cpu().numpy())
    plt.axis("off")
    plt.title("Generated Images After Training")
    plt.show()

# Display the 32 generated images using fixed noise
with torch.no_grad():
    fake_images = generator(fixed_noise).detach().cpu()

display_images(fake_images)
```

Generated Images After Training



In [20]:

```
# Plot for Generator and Discriminator Losses(after training)
plt.plot(gen_loss_profile, color='red', label='Generator Loss')
plt.plot(disc_loss_profile, color='blue', label='Discriminator Loss')
plt.legend()
plt.xlabel('Training steps')
plt.ylabel('Loss')
plt.title('Training Loss Profiles')
plt.show()
```

In [21]:

```
!pip install pytorch-fid
import os
from torchvision.utils import save_image
import subprocess

# Create folders for real and fake images
os.makedirs("fid_images/real", exist_ok=True)
os.makedirs("fid_images/fake", exist_ok=True)

# Save 1000 real images
real_count = 0
for real_batch, _ in test_loader:
    for img in real_batch:
        if real_count >= 1000:
            break
        save_image(img, f"fid_images/real/{real_count}.png", normalize=True)
        real_count += 1
    if real_count >= 1000:
        break

# Save 1000 fake images
generator.eval()
with torch.no_grad():
    fake_count = 0
    while fake_count < 1000:
        z = torch.randn(batch_size, z_dim).to(device)
        fake_batch = generator(z).detach().cpu()
        for img in fake_batch:
```

```
            if fake_count >= 1000:
                break
            save_image(img, f"fid_images/fake/{fake_count}.png", normalize=T
            fake_count += 1
```

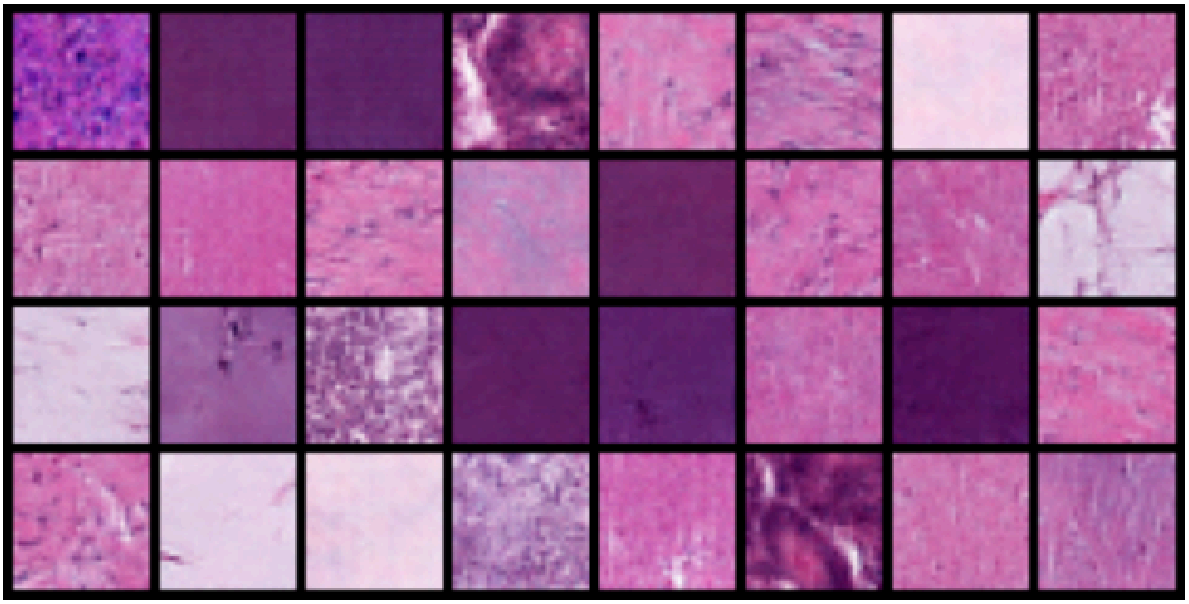
```
Collecting pytorch-fid
  Downloading pytorch_fid-0.3.0-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packa
ges (from pytorch-fid) (1.26.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-pack
ages (from pytorch-fid) (11.1.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packa
ges (from pytorch-fid) (1.15.2)
Requirement already satisfied: torch>=1.0.1 in /usr/local/lib/python3.11/dis
t-packages (from pytorch-fid) (2.5.1+cu124)
Requirement already satisfied: torchvision>=0.2.2 in /usr/local/lib/python3.
11/dist-packages (from pytorch-fid) (0.20.1+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
ckages (from torch>=1.0.1->pytorch-fid) (3.18.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/py
thon3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (4.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pa
ckages (from torch>=1.0.1->pytorch-fid) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pack
ages (from torch>=1.0.1->pytorch-fid) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-pack
ages (from torch>=1.0.1->pytorch-fid) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/lo
cal/lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/li
b/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/li
b/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/li
b/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/
lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/
lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/loca
l/lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.3.1.170)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/py
thon3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/loca
l/lib/python3.11/dist-packages (from torch>=1.0.1->pytorch-fid) (12.4.127)
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/di
st-packages (from torch>=1.0.1->pytorch-fid) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/di
st-packages (from torch>=1.0.1->pytorch-fid) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
11/dist-packages (from sympy==1.13.1->torch>=1.0.1->pytorch-fid) (1.3.0)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.11/dist-pac
kages (from numpy->pytorch-fid) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.11/dist-
packages (from numpy->pytorch-fid) (1.2.4)
```

```
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.11/dist-p
ackages (from numpy->pytorch-fid) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-package
s (from numpy->pytorch-fid) (2025.1.0)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/dist-pack
ages (from numpy->pytorch-fid) (2022.1.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.11/dist
-packages (from numpy->pytorch-fid) (2.4.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/
dist-packages (from jinja2->torch>=1.0.1->pytorch-fid) (3.0.2)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/local/lib/py
thon3.11/dist-packages (from mkl->numpy->pytorch-fid) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.11/dist
-packages (from mkl->numpy->pytorch-fid) (2022.1.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.11/dist
-packages (from tbb==2022.*->mkl->numpy->pytorch-fid) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.
11/dist-packages (from mkl_umath->numpy->pytorch-fid) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/li
b/python3.11/dist-packages (from intel-openmp<2026,>=2024->mkl->numpy->pytor
ch-fid) (2024.2.0)
Downloading pytorch_fid-0.3.0-py3-none-any.whl (15 kB)
Installing collected packages: pytorch-fid
Successfully installed pytorch-fid-0.3.0
```

In [22]:

```
import torch

# 32 samples using random noise
generator.eval()
with torch.no_grad():
    test_samples_final = torch.randn(32, z_dim).to(device)
    generated_images_final = generator(test_samples_final).detach().cpu()


# Display the 32 generated images using random noise
display_images(generated_images_final)
```

Generated Images After Training



In [23]:
```
# Compute FID score
fid_score = subprocess.check_output("pytorch-fid fid_images/real fid_images/
print("FID Score:", fid_score)
```

```
Downloading: "https://github.com/mseitzer/pytorch-fid/releases/download/fid_
weights/pt_inception-2015-12-05-6726825d.pth" to /root/.cache/torch/hub/chec
kpoints/pt_inception-2015-12-05-6726825d.pth
100%|██████████| 91.2M/91.2M [00:01<00:00, 79.7MB/s]
100%|██████████| 20/20 [00:02<00:00,  7.89it/s]
100%|██████████| 20/20 [00:02<00:00,  8.97it/s]
FID Score: FID:  102.12400348153452
```

Explanation:

- The above Deep Convolutional GAN (DCGAN) on the PathMNIST dataset has a generator architecture based on the given ConvTranspose2D block diagram.
- The discriminator used 5 convolutional layers with LeakyReLU activations and BatchNorm. The model was trained for up to 1000 epochs with early stopping (patience=40), which trigerred at epoch 56.
- Generator and discriminator losses showed stable trends with no evidence of mode collapse.
- The computed FID score used 1000 real and 1000 generated images, obtaining a score of 102.12.
- Visual inspection of generated images showed variety in texture and color, confirming the absence of mode collapse.

Hyperparameter used:

Hyperparameter	Generator	Discriminator
Activation Function (Hidden)	ReLU	LeakyReLU(0.2)
Activation Function (Output)	tanh (to scale output to [-1, 1])	None (raw logits passed to BCE loss)
Weight Initializer	PyTorch default (Kaiming/He)	PyTorch default (Kaiming/He)
Number of Hidden Layers	4 (Linear + 3 ConvTranspose2d)	5 (Conv2d layers)
Neurons in Hidden Layers	1024 → 512 → 256 → 128	64 → 128 → 256 → 512 → 1024
Loss Function	BCEWithLogitsLoss() (target = 1 for fake)	BCEWithLogitsLoss() (target = 1 for real, 0 for fake)
Optimizer	Adam	Adam
Learning Rate	0.0002	0.0002
Batch Size	64	64
Number of Epochs	1000 epochs (with patience=40 & early stopping)	1000 epochs (with patience=40 & early stopping)
Evaluation Metric	FID Score (1000 fake images generated)	FID Score (compared to 1000 real images)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: