Github Repo

https://github.com/Deep-Learning-Group-B/Case-Study-2

In [ ]:
```
import os
import json

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

from tqdm import tqdm_notebook

import tensorflow as tf
from tensorflow.keras.layers import Dense, Reshape, Flatten, Dropout, LeakyReLU
from tensorflow.keras.layers import Conv2D, Conv2DTranspose
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_arra

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

Load processed data

In [ ]:
```
# Loading images from file
images = np.load('/content/drive/MyDrive/sem3_DL/images.npy')

# Loading labels from file
labels = np.load('/content/drive/MyDrive/sem3_DL/labels.npy')

print(f"Images loaded: {images.shape}")
print(f"Labels loaded: {labels.shape}")
```

```
Images loaded: (8390, 224, 224, 3)
Labels loaded: (8390,)
```

In [ ]:
```
labels_unique = np.unique(labels)
print(f"Unique labels: {labels_unique}")
```

```
Unique labels: ['drink' 'food' 'inside' 'menu' 'outside']
```

In [ ]:
```
# Separate images by class
selected_class = 'food'  # Replace with the actual class label you want to use
class_indices = np.where(labels == selected_class)[0]
class_images = images[class_indices]

print(f"Total images for class '{selected_class}': {class_images.shape}")
```

```
Total images for class 'food': (1678, 224, 224, 3)
```

In [ ]:
```
# Plot the images
num_images = 5
fig, axs = plt.subplots(1, num_images, figsize=(15, 15))
```

```
for i in range(num_images):
    axs[i].imshow(class_images[i, :, :, :])  # Display RGB images
    axs[i].axis('off')
plt.show()
```

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

In [ ]:
```
# Generator
def build_generator():
    model = Sequential()
    model.add(Dense(512 * 14 * 14, activation="relu", input_dim=100))
    model.add(Reshape((14, 14, 512)))
    model.add(Conv2DTranspose(256, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Conv2DTranspose(3, kernel_size=4, strides=2, activation="tanh", pa
    return model

# Discriminator
def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=4, strides=2, input_shape=(224, 224, 3), pa
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Conv2D(128, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Conv2D(256, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

In [ ]:
```
# Compile the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['ac

# Build and compile the combined model
generator = build_generator()
z = tf.keras.Input(shape=(100,))
```

```
img = generator(z)
discriminator.trainable = False
valid = discriminator(img)
combined = tf.keras.Model(z, valid)
combined.compile(loss='binary_crossentropy', optimizer=Adam())
```

In [ ]:
```
generator.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape
dense_1 (Dense)	(None, 100352)
reshape (Reshape)	(None, 14, 14, 512)
conv2d_transpose (Conv2DTranspose)	(None, 28, 28, 256)
leaky_re_lu_3 (LeakyReLU)	(None, 28, 28, 256)
conv2d_transpose_1 (Conv2DTranspose)	(None, 56, 56, 128)
leaky_re_lu_4 (LeakyReLU)	(None, 56, 56, 128)
conv2d_transpose_2 (Conv2DTranspose)	(None, 112, 112, 64)
leaky_re_lu_5 (LeakyReLU)	(None, 112, 112, 64)
conv2d_transpose_3 (Conv2DTranspose)	(None, 224, 224, 3)

```
Total params: 12,891,587 (49.18 MB)
Trainable params: 12,891,587 (49.18 MB)
Non-trainable params: 0 (0.00 B)
```

In [ ]:
```
discriminator.summary()
```

Model: "sequential"

```
┃ Layer (type)                     ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
│ conv2d (Conv2D)                  │ (None, 112, 112, 64)        │
├──────────────────────────────────┼─────────────────────────────┼
│ leaky_re_lu (LeakyReLU)          │ (None, 112, 112, 64)        │
├──────────────────────────────────┼─────────────────────────────┼
│ dropout (Dropout)                │ (None, 112, 112, 64)        │
├──────────────────────────────────┼─────────────────────────────┼
│ conv2d_1 (Conv2D)                │ (None, 56, 56, 128)         │
├──────────────────────────────────┼─────────────────────────────┼
│ leaky_re_lu_1 (LeakyReLU)        │ (None, 56, 56, 128)         │
├──────────────────────────────────┼─────────────────────────────┼
│ dropout_1 (Dropout)              │ (None, 56, 56, 128)         │
├──────────────────────────────────┼─────────────────────────────┼
│ conv2d_2 (Conv2D)                │ (None, 28, 28, 256)         │
├──────────────────────────────────┼─────────────────────────────┼
│ leaky_re_lu_2 (LeakyReLU)        │ (None, 28, 28, 256)         │
├──────────────────────────────────┼─────────────────────────────┼
│ dropout_2 (Dropout)              │ (None, 28, 28, 256)         │
├──────────────────────────────────┼─────────────────────────────┼
│ flatten (Flatten)                │ (None, 200704)              │
├──────────────────────────────────┼─────────────────────────────┼
│ dense (Dense)                    │ (None, 1)                   │
└──────────────────────────────────┴─────────────────────────────┴

 Total params: 859,585 (3.28 MB)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 859,585 (3.28 MB)
```

In [ ]:
```python
combined.summary()
```

```
Model: "functional_20"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
┃ Layer (type)                     ┃ Output Shape                ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
│ input_layer_2 (InputLayer)       │ (None, 100)                 │
├──────────────────────────────────┼─────────────────────────────┼
│ sequential_1 (Sequential)        │ (None, 224, 224, 3)         │
├──────────────────────────────────┼─────────────────────────────┼
│ sequential (Sequential)          │ (None, 1)                   │
└──────────────────────────────────┴─────────────────────────────┴

 Total params: 13,751,172 (52.46 MB)
 Trainable params: 12,891,587 (49.18 MB)
 Non-trainable params: 859,585 (3.28 MB)
```

In [ ]:
```python
# Function to generate and plot images
def generate_and_plot_images(generator, num_images=5):
    # Generate noise and use the generator to create images
    noise = np.random.normal(0, 1, (num_images, 100))
    generated_images = generator.predict(noise)

    # Rescale images 0 - 1
    generated_images = 0.5 * generated_images + 0.5
    # Plot the images
    fig, axs = plt.subplots(1, num_images, figsize=(15, 15))
    for i in range(num_images):
        axs[i].imshow(generated_images[i, :, :, :])  # Display RGB images
        axs[i].axis('off')
    plt.show()
```

In [ ]:

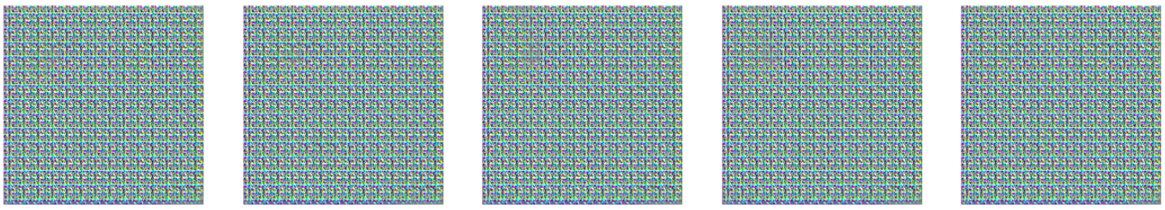
```
# Training parameters
epochs = 100
batch_size = 64
save_interval = 50

# Training Loop
for epoch in range(epochs):
    # Train Discriminator
    idx = np.random.randint(0, class_images.shape[0], batch_size)
    real_imgs = class_images[idx]
    noise = np.random.normal(0, 1, (batch_size, 100))
    fake_imgs = generator.predict(noise)
    d_loss_real = discriminator.train_on_batch(real_imgs, np.ones((batch_size, 1
    d_loss_fake = discriminator.train_on_batch(fake_imgs, np.zeros((batch_size,
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Train Generator
    noise = np.random.normal(0, 1, (batch_size, 100))
    g_loss = combined.train_on_batch(noise, np.ones((batch_size, 1)))

    # Print the progress
    if epoch % save_interval == 0:
        print(f"{epoch} [D loss: {d_loss[0]}] [G loss: {g_loss}]")
        generate_and_plot_images(generator, num_images=5)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
0 [D loss: 0.6547226309776306] [G loss: [array(0.6675807, dtype=float32), array
(0.6675807, dtype=float32), array(0.4765625, dtype=float32)]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 892ms/step
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_
train_function.<locals>.one_step_on_iterator at 0x7def680ed7e0> triggered tf.func
tion retracing. Tracing is expensive and the excessive number of tracings could b
e due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_re
tracing=True option that can avoid unnecessary retracing. For (3), please refer t
o https://www.tensorflow.org/guide/function#controlling_retracing and https://ww
w.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_
train_function.<locals>.one_step_on_iterator at 0x7def680efc70> triggered tf.func
tion retracing. Tracing is expensive and the excessive number of tracings could b
e due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_re
tracing=True option that can avoid unnecessary retracing. For (3), please refer t
o https://www.tensorflow.org/guide/function#controlling_retracing and https://ww
w.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
50 [D loss: 0.9829340577125549] [G loss: [array(0.9862858, dtype=float32), array
(0.9862858, dtype=float32), array(0.45909926, dtype=float32)]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

In [ ]:

```
generator.save('/content/drive/MyDrive/sem3_DL/deployment/DCGAN_generator_food.h
discriminator.save('/content/drive/MyDrive/sem3_DL/deployment/DCGAN_discriminato
combined.save('/content/drive/MyDrive/sem3_DL/deployment/DCGAN_combined_food.h5'
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or `ke
ras.saving.save_model(model, 'my_model.keras')`.
```

In [ ]: