In [64]:
```
# Import necessary libraries
import gradio as gr
import numpy as np
from tensorflow.keras.models import load_model
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import torch
from PIL import Image
import pickle
from tensorflow.keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import get_custom_objects
```

Creating custom object for LinformerSelfAttention

In [ ]:
```
from tensorflow.keras.layers import Dense, Dropout, LayerNormalization
from tensorflow.keras.regularizers import l2
import tensorflow as tf
from tensorflow.keras.layers import Layer, Dense, Dropout
from linformer import Linformer

class LinformerSelfAttention(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, seq_len, dropout_rate=0.4, key_dim=Non
        super().__init__(**kwargs)
        super(LinformerSelfAttention, self).__init__(**kwargs)  # Pass kwargs to th
        self.num_heads = num_heads
        self.key_dim = key_dim
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.seq_len = seq_len
        self.key_dim = key_dim

        self.attention = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.key_projection = Dense(embed_dim)  # Project keys to the same dimensio
        self.value_projection = Dense(embed_dim)  # Project values to the same dime
        self.dropout = Dropout(dropout_rate)
        self.layernorm = LayerNormalization()

    def get_config(self):
        config = super().get_config()
        config.update({
            'embed_dim': self.embed_dim,
            'num_heads': self.num_heads,
            'seq_len': self.seq_len,
            'dropout_rate': self.dropout_rate,
            'key_dim': self.key_dim  # Add key_dim to config
        })
        return config

    def call(self, inputs):
        # Project keys and values to match the dimension of queries
        keys = self.key_projection(inputs)
        values = self.value_projection(inputs)
```

```
        # Compute multi-head attention with projected keys and values
        attention_output = self.attention(inputs, keys, values)
        attention_output = self.dropout(attention_output)

        # Residual connection and layer normalization
        return self.layernorm(inputs + attention_output)


def Linformer_block(inputs, embed_dim, num_heads, ff_dim, seq_len, dropout_rate=0.4
    attention_output = LinformerSelfAttention(embed_dim, num_heads, seq_len, dropou

    # Feed-forward layer with increased L2 regularization
    ffn_output = Dense(ff_dim, activation="relu", kernel_regularizer=l2(0.02))(atte
    ffn_output = Dense(embed_dim, kernel_regularizer=l2(0.02))(ffn_output)
    ffn_output = Dropout(dropout_rate)(ffn_output)

    # Add & Norm
    return LayerNormalization()(attention_output + ffn_output)
```

Loading all 6 Models

DCGAN, CGAN, CNN, TranferLearnet Model and 2 transformers for sentiment analysis

In [177…

```
# Image generation models
dcgan_model = load_model("models/DCGAN_generator_food.h5")
cgan_model = load_model("models/generator_model.h5")

# Image classification models
cnn_model = load_model("models/CNN_Model.h5")
transfer_model = load_model("models/EffecientNet_TransferModel.h5")

# Sentiment analysis models
# Load the tokenizer
with open('models/tokenizer.pkl', 'rb') as f:
    tokenizer = pickle.load(f)

sentiment_model_1 = load_model("models/causal_transformer.keras")

custom_objects = {
    'LinformerSelfAttention': LinformerSelfAttention,
    'Linformer_block': Linformer_block
}
# Load the model
sentiment_model_2 = load_model('models/linformer_model.keras', custom_objects=custo

'''custom_objects = {'LinformerSelfAttention': LinformerSelfAttention}
sentiment_model_2 = load_model('models/linformer_model.h5', custom_objects=custom_o
```

```
WARNING:absl:No training configuration found in the save file, so the model was *not
* compiled. Compile it manually.
WARNING:absl:No training configuration found in the save file, so the model was *not
* compiled. Compile it manually.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be buil
t. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be buil
t. `model.compile_metrics` will be empty until you train or evaluate the model.
c:\Users\janvi\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\s
aving\saving_lib.py:719: UserWarning: Skipping variable loading for optimizer 'rmspr
op', because it has 69 variables whereas the saved optimizer has 136 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
c:\Users\janvi\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\l
ayers\layer.py:391: UserWarning: `build()` was called on layer 'linformer_self_atten
tion_42', however the layer does not have a `build()` method implemented and it look
s like it has unbuilt state. This will cause the layer to be marked as built, despit
e not being actually built, which may cause failures down the line. Make sure to imp
lement a proper `build()` method.
  warnings.warn(
c:\Users\janvi\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\l
ayers\layer.py:391: UserWarning: `build()` was called on layer 'linformer_self_atten
tion_44', however the layer does not have a `build()` method implemented and it look
s like it has unbuilt state. This will cause the layer to be marked as built, despit
e not being actually built, which may cause failures down the line. Make sure to imp
lement a proper `build()` method.
  warnings.warn(
```

Out[177…

```
"custom_objects = {'LinformerSelfAttention': LinformerSelfAttention}\nsentiment_mo
del_2 = load_model('models/linformer_model.h5', custom_objects=custom_objects)"
```

Function for image generation for DCGAN

In [62]:

```
#Function for image generation
def generate_images(model, num_images=1):
    generated_images = []
    for _ in range(num_images):
        noise = tf.random.normal([1, 100])  # Adjust according to your model's inpu
        image = model(noise)
        image = (image + 1) / 2  # Normalize to [0, 1]
        image = tf.squeeze(image)  # Use tf.squeeze() for TensorFlow tensors
        generated_images.append(image.numpy())  # Convert to numpy array for Gradio
    return generated_images


    '''noise = np.random.normal(0, 1, (5, 100))
    generated_images = model.predict(noise)

    # Rescale images 0 - 1
    generated_images = 0.5 * generated_images + 0.5

    images = generate_images(model=dcgan_model)
    return images[0]'''
```

Function for image generation for CGAN

In [200…

```
c_labels = ['food', 'outside', 'drink', 'inside', 'menu']

def generate_cond_images(num_images=5, selected_label="food"):
    generated_images = []

    # Step 1: Get the index of the selected label
    label_index = c_labels.index(selected_label)  # Find the index of the selected
    label = np.zeros((1, len(c_labels)))  # One-hot encoded label vector
    label[0, label_index] = 1  # Set the corresponding label to 1
    label = tf.convert_to_tensor(label, dtype=tf.float32)  # Convert label to tenso

    # Step 2: Ensure the label is properly formatted as a tensor
    label = tf.cast(label, tf.float32)  # Ensure it is of type float32

    for _ in range(num_images):
        # Step 3: Generate random noise vector with shape (1, 200) instead of (1, 1
        noise = tf.random.normal([1, 200])  # Adjusted noise shape to match the mod

        # Step 4: Ensure both noise and label are tensors
        noise = tf.cast(noise, tf.float32)  # Ensure noise is a tensor of type floa

        # Step 5: Generate image using both noise and the selected label
        image = cgan_model([noise, label])  # Pass both noise and label to the mode
        image = (image + 1) / 2  # Normalize to [0, 1]
        image = tf.squeeze(image)  # Remove extra dimensions (if any)

        generated_images.append(image.numpy())  # Convert tensor to numpy array

    return generated_images
```

Function to classify image using CNN

In [93]:

```
import numpy as np
from PIL import Image
import gradio as gr
from tensorflow.keras.preprocessing.image import img_to_array

# Example function to handle image resizing and classification
def classify_image(image, model):
    # Convert numpy array to PIL Image if it's in numpy array format
    if isinstance(image, np.ndarray):
        image = Image.fromarray(image)

    # Resize the image to (128, 128)
    img = image.resize((128, 128))  # Resize the image using PIL's resize method

    # Convert the image to an array
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Normalize the image (assuming your model expects pixel values in [0, 1] range
    img_array = img_array / 255.0

    # Make predictions with the model
    predictions = model.predict(img_array)
```

```
    # Define the labels corresponding to your model's output
    labels = ['food', 'outside', 'drink', 'inside', 'menu']

    # Return the prediction as a dictionary with labels
    return {labels[i]: float(predictions[0][i]) for i in range(len(labels))}
```

FUnction to classify image using Transfer Learning

In [126…

```
import numpy as np
import gradio as gr
from PIL import Image

def transfer_classify_image(image, model):
    # Preprocess the image (resize, normalize, etc.)
    img = image.resize((224, 224))  # Resize to the expected input size (224x224)
    img_array = np.array(img)  # Convert to array
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = img_array / 255.0  # Normalize image (assuming the model was traine

    # Ensure that the image data has the correct shape
    if img_array.shape[0] != 1:
        raise ValueError(f"Image batch size mismatch: expected 1, got {img_array.sh

    # Pass the image to the model
    predictions = model.predict(img_array)

    # Assume you have labels for classification (modify as per your model output)
    labels = ['food', 'outside', 'drink', 'inside', 'menu']

    # Return the predictions as a dictionary with labels
    return {labels[i]: float(predictions[0][i]) for i in range(len(labels))}
```

Function for sentiment Analysis

In [ ]:

```
def preprocess_text(text):
    # Tokenize the incoming text
    seq = tokenizer.texts_to_sequences([text])
    seq = pad_sequences(seq, maxlen=200, padding='post', truncating='post')
    return seq

#Function for sentiment analysis
def sentiment_analysis(text, model):
    preprocessed_text = preprocess_text(text)

    # Predict using the model (assuming the model is a Keras model)
    prediction = model.predict(preprocessed_text)  # If your model has multiple out

    # If your model outputs logits, apply softmax to get probabilities
    probs = prediction[0]  # Assuming the model output shape is (batch_size, num_cl

    # Define possible labels (adjust based on your specific task)
    labels = ["Negative", "Positive", "Neutral"]

    # Return the probability for each label
```

```
    return {labels[i]: float(probs[i]) for i in range(len(labels))}

    #return {labels[i]: float(probs[0][i]) for i in range(len(labels))}
```

Generating Interfaces for all 6 models

In [ ]:

```
#Gradio Interfaces

def generate_images_with_model(num_images):
    return generate_images(dcgan_model, num_images)

dcgan_interface = gr.Interface(
    fn=generate_images_with_model,
    inputs=gr.Slider(minimum=1, maximum=10, step=1, value=5, label="Number of Image
    outputs=gr.Gallery(type="numpy", label="Generated Images"),
    title="DCGAN Image Generator",
    description="Generates images using a DCGAN model. Adjust the slider to choose
)

cgan_interface = gr.Interface(
    fn=generate_cond_images,  # Function to call
    inputs=[
        gr.Slider(minimum=1, maximum=10, step=1, value=5, label="Number of Images")
        gr.Dropdown(choices=c_labels, label="Select Label", value="food")  # Dropdo
    ],
    outputs=gr.Gallery(label="Generated Images"),  # Display images in a gallery fo
    title="CGAN Image Generator",  # Title of the interface
    description="Generates images using a CGAN model. Adjust the slider to choose t
)


cnn_interface = gr.Interface(
    fn=lambda image: classify_image(image=image, model=cnn_model),
    inputs=gr.Image(type="numpy", label="Upload Image"),  # Input should be an uplo
    outputs=gr.Label(label="Predicted Label"),
    title="CNN Image Classifier",
    description="Classifies an image using a CNN model."
)

transfer_interface = gr.Interface(
    fn=lambda image: transfer_classify_image(image=image, model=transfer_model),
    inputs=[gr.Image(type="pil", label="Upload Image")],  # Input for the image
    outputs=gr.Label(label="Predicted Label"),
    title="Transfer Learning Image Classifier",
    description="Classifies an image using a transfer learning model."
)


# Transformer Sentiment Analysis Interfaces
sentiment_1_interface = gr.Interface(
    fn=lambda text: sentiment_analysis(text= text, model=sentiment_model_1),
    inputs=gr.Textbox(label="Input Text"),
    outputs=gr.Label(num_top_classes=3),
    title="Transformer Sentiment Analysis (Model 1)",
```

```
    description="Predicts sentiment of a review using Transformer Model 1."
)

sentiment_2_interface = gr.Interface(
    fn=lambda text: sentiment_analysis(text= text, model=sentiment_model_1),
    inputs=gr.Textbox(label="Input Text"),
    outputs=gr.Label(num_top_classes=3),
    title="Transformer Sentiment Analysis (Model 1)",
    description="Predicts sentiment of a review using Transformer Model 1.",
)
```

Out[ ]:
```
'sentiment_2_interface = gr.Interface(\n    fn=lambda text: sentiment_analysis(tex
t= text, model=sentiment_model_2),\n    inputs=gr.Textbox(label="Input Text"),\n
outputs=gr.Label(num_top_classes=3),\n    title="Transformer Sentiment Analysis (M
odel 2)",\n    description="Predicts sentiment of a review using Transformer Model
2."\n)'
```

In [ ]:
```
app = gr.TabbedInterface(
    [dcgan_interface, cgan_interface, cnn_interface, transfer_interface, sentiment_
    tab_names=["DCGAN", "CGAN","CNN Classifier", "Transfer Learning", "Sentiment 1"
)

# Launch the app
app.launch(share=True)
```

```
* Running on local URL:  http://127.0.0.1:7912
* Running on public URL: https://57a7c64d755ab5cbf4.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from the terminal in the working directory to deploy to Hugging Fa
ce Spaces (https://huggingface.co/spaces)
```

DCGAN CGAN CNN Classifier Transfer Learning •••

DCGAN Image Generator

Generates images using a DCGAN model. Adjust the slider to choose the number of images to generate.

Number of Images 5 ↺

1 10

Clear Submit

Generated Images

Out[ ]:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
WARNING:tensorflow:5 out of the last 12 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002BB92F2E480> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be 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_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```


```
WARNING:tensorflow:5 out of the last 12 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000002BB92F2E480> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be 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_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 294ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

In [ ]: