UNIVERSIDAD
DE ALMERÍA

SOUND TO SOMETHING

Transformación de Audio en Contenido Visual mediante IA Generativa

Augustin Alexandru Besu

Mayo 2025

Índice

Índice de figuras

Listings

1. Introducción

1.1. Contexto del Proyecto

El presente proyecto tiene como objetivo principal la implementación de un sistema de generación de imágenes a partir de archivos de audio, tal como indica claramente el título *Sound to Something*. Esta propuesta surge del interés por explorar las posibilidades que ofrece la inteligencia artificial moderna para establecer conexiones automáticas entre modalidades sensoriales diferentes, específicamente entre el dominio auditivo y el visual.

1.2. Planteamiento Metodológico

La conversión directa de contenido musical a imágenes implica la comprensión y procesamiento de información en varios niveles de abstracción. En primer lugar, es necesario analizar las características acústicas del audio de entrada para extraer información relevante sobre el género musical, las características espectrales y los patrones rítmicos. Posteriormente, esta información debe ser interpretada y traducida a un lenguaje comprensible para los modelos de generación visual, lo que requiere un mapeo semántico entre conceptos musicales y descriptores visuales.

Desarrollar cada uno de estos componentes desde cero supondría una cantidad de tiempo ingente que excedería ampliamente el alcance temporal. Un enfoque completamente original requeriría la implementación de algoritmos de procesamiento de señales desde sus fundamentos matemáticos, el diseño y entrenamiento de arquitecturas de aprendizaje profundo especializadas para clasificación musical, y el desarrollo de modelos generativos propios para la síntesis de imágenes.

1.3. Estrategia de Implementación

Considerando estas limitaciones temporales y de alcance, se ha optado por un enfoque pragmático basado en la integración inteligente de tecnologías y metodologías existentes mediante un pipeline de procesamiento especializado. Esta estrategia permite concentrar los esfuerzos de desarrollo en los aspectos más críticos e innovadores del proyecto: el diseño del flujo de datos, el mapeo semántico entre características musicales y descriptores visuales, y la orquestación coherente de diferentes tecnologías para alcanzar el resultado esperado.

El pipeline resultante aprovecha bibliotecas especializadas para el procesamiento de audio, modelos preentrenados para la generación de imágenes, y frameworks modernos de aprendizaje automático. Esta aproximación no solo resulta más eficiente en términos de tiempo de desarrollo, sino que también garantiza la utilización de soluciones relativamente robustas, permitiendo así centrarse en la innovación arquitectónica y la integración de componentes heterogéneos.

1.4. Estructura de la Memoria

La presente memoria se organiza de manera progresiva para facilitar la comprensión del sistema desarrollado y las decisiones metodológicas adoptadas. Las secciones posteriores abordarán los fundamentos teóricos que sustentan el proyecto, el análisis detallado de cada componente del pipeline, los aspectos específicos de implementación y la evaluación de los resultados obtenidos. Esta estructura tiene como objetivo proporcionar al lector una comprensión completa del proyecto, desde la conceptualización inicial hasta la implementación final, preparando el contexto necesario para las secciones técnicas subsiguientes. Cabe destacar que este proyecto no concluye con esta entrega, sino que continuará evolucionando y mejorándose con el tiempo.

2. Desarrollo del Sistema

2.1. Consideraciones Tecnológicas y de Hardware

Una característica fundamental del sistema es su implementación basada en modelos de código abierto y gratuitos, específicamente el modelo *dreamlike-art/dreamlike-diffusion-1.0* para la generación de imágenes. Esta decisión metodológica responde al objetivo de desarrollar una solución accesible que no requiera el uso de servicios de pago como OpenAI DALL-E o Midjourney. Sin embargo, esta aproximacion implica que todo el procesamiento computacional debe ejecutarse localmente en el hardware del usuario.

Los requisitos de hardware representan uno de los aspectos más críticos del sistema. La generación de imágenes y videos mediante modelos de difusión es computacionalmente intensiva y requiere recursos sustanciales, particularmente en términos de memoria gráfica. Para el desarrollo y pruebas del proyecto se utilizó un sistema equipado con una tarjeta gráfica NVIDIA RTX 4070 Ti, 32 GB de memoria RAM y un procesador Intel i7-13700KF. Con esta configuración, la generación de una imagen típica requiere entre 30 segundos y 2 minutos, mientras que la generación de videos puede extenderse entre 5 y 20 minutos dependiendo de la resolución y duración solicitadas.

El sistema está diseñado para funcionar también con hardware más modesto, incluyendo la opción de utilizar únicamente CPU en ausencia de una tarjeta gráfica compatible con CUDA. No obstante, esta modalidad resulta en tiempos de procesamiento significativamente superiores, pudiendo requerir entre 10 y 30 minutos para la generación de una sola imagen. Esta flexibilidad en los requisitos de hardware garantiza la accesibilidad del sistema, aunque con diferencias considerables en la experiencia de usuario.

2.2. Entrenamiento del Clasificador de Géneros

Antes de abordar el pipeline principal de transformación, es fundamental describir el proceso de entrenamiento del clasificador de géneros musicales, ya que este componente constituye una de las piezas clave del sistema. A diferencia de otros elementos que utilizan modelos preentrenados, el clasificador de géneros representa una contribución original desarrollada específicamente para este proyecto.

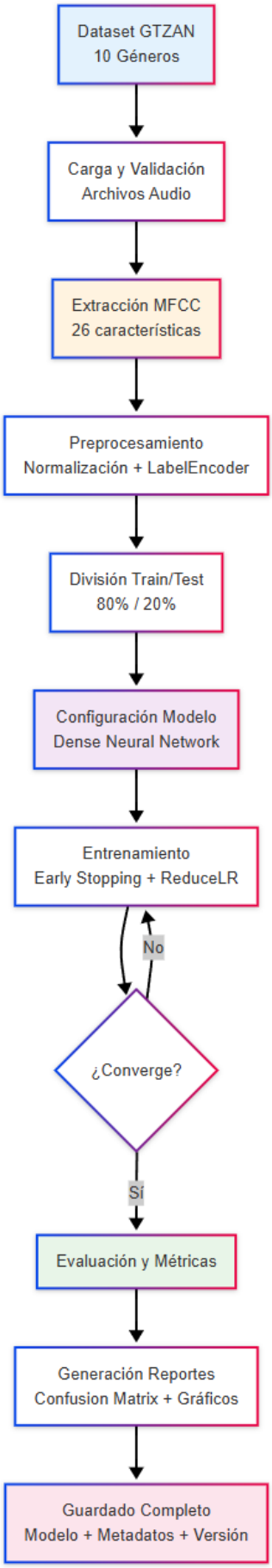

Figura 1: Pipeline de entrenamiento del clasificador de géneros musicales.

El proceso de entrenamiento, implementado en el módulo `audio_classifier_trainer.py`, utiliza el reconocido dataset GTZAN que contiene muestras de audio clasificadas en diez géneros musicales diferentes. Como se observa en la Figura 1, el entrenamiento sigue un flujo estructurado que incluye la extracción de características MFCC, el preprocesamiento de datos, la configuración de la arquitectura de red neuronal, y la optimización mediante técnicas de regularización y early stopping.

Una vez completado el entrenamiento, el sistema genera varios artefactos importantes: el modelo entrenado en formato H5, los encoders necesarios para el preprocesamiento, archivos de metadatos con información sobre la precisión alcanzada, y visualizaciones del proceso de entrenamiento incluyendo matrices de confusión y gráficos de evolución de la pérdida. Estos componentes forman la base sobre la cual opera el pipeline principal de transformación audio-visual.

2.3. Pipeline de Transformación Audio-Visual

Con el clasificador de géneros ya entrenado y disponible, el sistema principal puede proceder a la transformación de audio en contenido visual. El pipeline de transformación adopta un enfoque modular que facilita tanto el mantenimiento del código como la escalabilidad del sistema, permitiendo la separación clara de responsabilidades entre cada etapa del proceso de conversión.

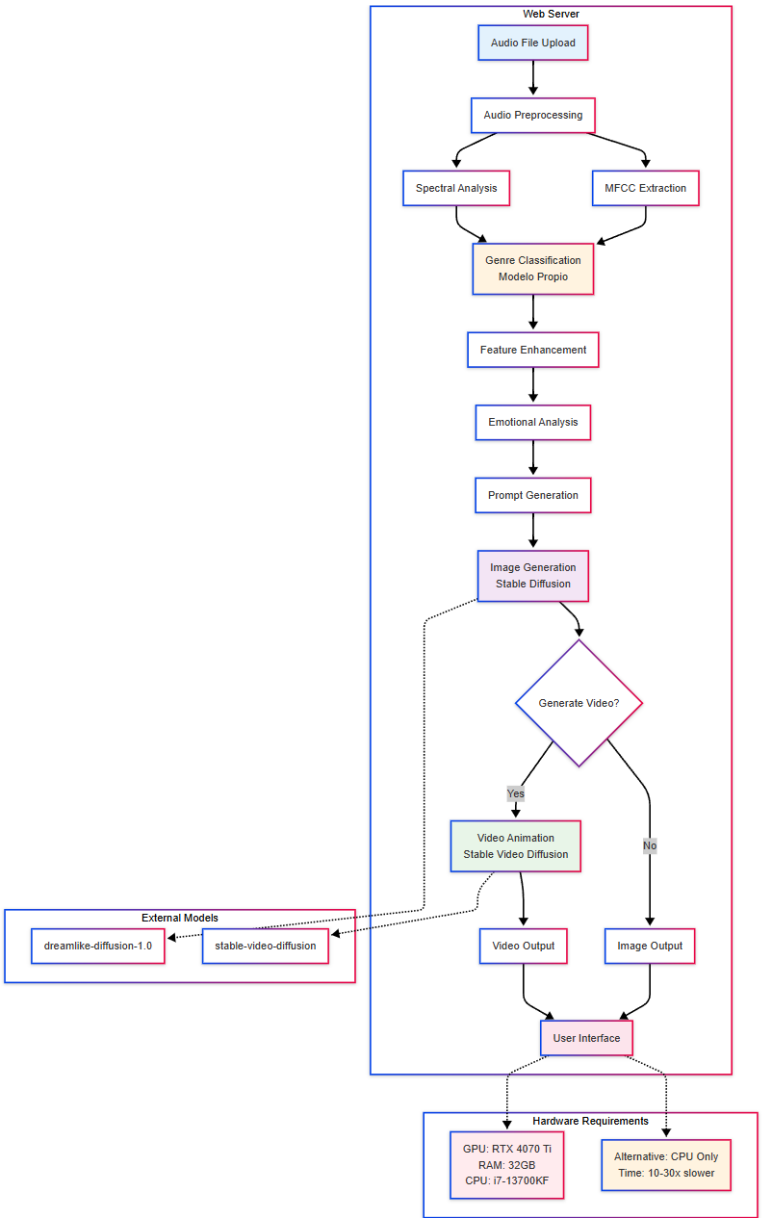

Figura 2: Pipeline de transformación de audio a contenido visual.

Como se observa en la Figura 2, el núcleo del sistema reside en el servidor web desarrollado con *Streamlit*, que proporciona una interfaz intuitiva para la interacción del usuario final. Esta implementación permite el acceso a todas las funcionalidades del sistema a través de un navegador web, eliminando la necesidad de instalaciones complejas por parte del usuario. Adicionalmente, el sistema mantiene la flexibilidad de ejecución directa mediante consola de Python, lo que resulta especialmente útil durante las fases de desarrollo, depuración y evaluación técnica del sistema.

El pipeline de procesamiento se estructura en cuatro fases principales claramente diferenciadas. La primera fase corresponde al análisis acústico del audio de entrada, donde se extraen las características MFCC y se realizan análisis espectrales y temporales que capturan la esencia matemática del contenido musical. La segunda fase carga y utiliza el clasificador de géneros previamente entrenado para identificar automáticamente el género musical del audio procesado. La tercera fase transforma estas características técnicas y la información de género en descriptores emocionales y estilísticos que enriquecen la comprensión del contenido musical. Finalmente, la cuarta fase utiliza esta información procesada para generar descripciones textuales optimizadas que guían los modelos de difusión en la síntesis del contenido visual correspondiente.

Esta arquitectura modular no solo simplifica las tareas de mantenimiento y mejora del sistema, sino que también permite optimizar cada componente de manera independiente, adaptándolo a las capacidades del hardware disponible. Al mantener una separación definida entre el análisis de audio, la clasificación de géneros y la generación visual, se asegura que cualquier futura actualización o mejora en uno de estos módulos pueda integrarse sin interferir con el funcionamiento del resto del sistema.

3. Coeficientes MFCC

Antes de adentrarnos en la implementación concreta del proyecto, es fundamental comprender uno de los pilares en los que se basa el procesamiento del audio en este sistema: los Coeficientes Cepstrales en las Frecuencias de Mel (MFCC)..

3.1. Introducción a los Coeficientes MFCC

Los Coeficientes Cepstrales en las Frecuencias de Mel (MFCC, por sus siglas en inglés) constituyen una de las técnicas más fundamentales y ampliamente utilizadas en el procesamiento de señales de audio y reconocimiento automático del habla [Davis and Mermelstein, 1980]. Esta técnica de extracción de características se basa en la percepción auditiva humana y ha demostrado ser extraordinariamente efectiva para capturar la información espectral más relevante de las señales sonoras.

El desarrollo de los MFCC se fundamenta en el reconocimiento de que el sistema auditivo humano no procesa las frecuencias de manera lineal, sino que muestra mayor sensibilidad a los cambios en las frecuencias bajas que en las altas [Stevens et al., 1937]. Esta característica perceptual se modela matemáticamente mediante la escala de Mel, que establece una correspondencia no lineal entre la frecuencia física y la frecuencia perceptual.

3.2. Proceso de Cálculo de los MFCC

El cálculo de los coeficientes MFCC implica una secuencia de transformaciones matemáticas que convierten la señal de audio temporal en un conjunto de características espectrales compactas y discriminativas. Este proceso se estructura en varias etapas claramente diferenciadas que se ilustran en la Figura 3.

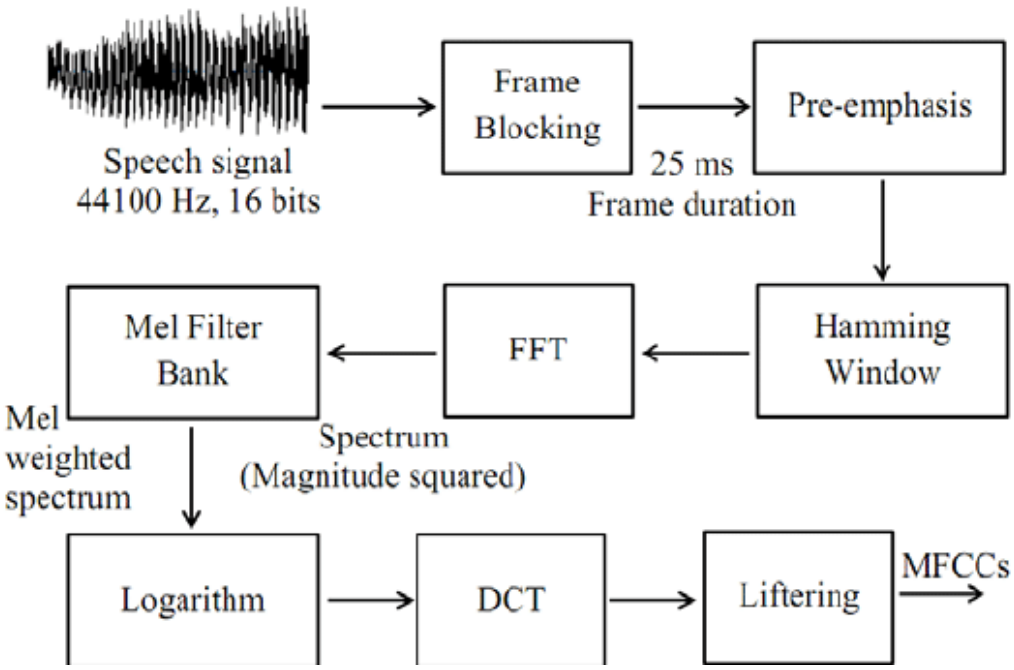

Figura 3: Proceso completo de extracción de coeficientes MFCC desde la señal de audio original.

La primera etapa consiste en la aplicación de una ventana temporal deslizante sobre la señal de audio, típicamente utilizando ventanas de Hamming o Hanning para minimizar los efectos de discontinuidad en los bordes. Posteriormente, se aplica la Transformada Rápida de Fourier (FFT) a cada ventana para obtener el espectro de frecuencias de la señal [Rabiner and Juang, 1993].

La tercera etapa implementa el filtrado mediante un banco de filtros triangulares distribuidos según la escala de Mel. Esta distribución no uniforme concentra mayor resolución en las frecuencias bajas, donde el oído humano presenta mayor discriminación, mientras que reduce la resolución en frecuencias altas donde la percepción es menos sensible [Mermelstein, 1976].

La cuarta etapa aplica el logaritmo a la energía de salida de cada filtro del banco de filtros Mel. Esta transformación logarítmica es crucial ya que comprime el rango dinámico de las energías espectrales y aproxima la respuesta no lineal del sistema auditivo humano a la intensidad sonora. El resultado es un espectro en escala logarítmica que refleja mejor la percepción humana del volumen.

Finalmente, la quinta etapa calcula la Transformada Discreta del Coseno (DCT) sobre los logaritmos de las energías de los filtros Mel. Esta transformación final produce los coeficientes MFCC propiamente dichos, decorrelacionando las características espectrales y concentrando la información más relevante en los primeros coeficientes [Sakar and Kursun, 2013]. Típicamente se utilizan los primeros 12 o 13 coeficientes MFCC, descartando el resto que contienen principalmente información de ruido o detalles espectrales menos relevantes para la clasificación.

3.3. Relevancia de los MFCC para Clasificación Musical

En el contexto específico de la clasificación de géneros musicales, los coeficientes MFCC aportan ventajas significativas que justifican su selección como técnica de extracción de características principal. Su capacidad para capturar tanto el contenido espectral como las características tímbricas de los instrumentos musicales los convierte en descriptores especialmente apropiados para esta tarea [Tzanetakis and Cook, 2002].

Los MFCC han demostrado efectividad particular en la distinción entre géneros musicales debido a su sensibilidad a las diferencias en instrumentación, técnicas de producción y características armónicas que definen cada estilo musical [Li et al., 2003]. Estudios comparativos han mostrado que los MFCC,

especialmente cuando se combinan con sus derivadas temporales, proporcionan tasas de clasificación superiores a otras técnicas de caracterización espectral en tareas de reconocimiento de género musical.

4. Implementación del sistema Sound to Something

En esta sección se aborda la implementación práctica del sistema desarrollado, examinando en detalle los componentes de software que hacen posible la transformación de audio en contenido visual. El análisis se estructura en subsecciones que cubren desde el entrenamiento del clasificador de géneros hasta la interfaz de usuario final, proporcionando una visión comprensiva de las decisiones de diseño, las técnicas implementadas y las consideraciones técnicas que sustentan cada módulo del sistema.

4.1. Módulo de Entrenamiento del Clasificador

El módulo de entrenamiento del clasificador, implementado en `audio_classifier_trainer.py`, constituye el componente responsable de generar el modelo de clasificación de géneros musicales que posteriormente utiliza el sistema principal. Este módulo está diseñado como una herramienta independiente que se ejecuta mediante línea de comandos, proporcionando al usuario control completo sobre el proceso de entrenamiento a través de una interfaz interactiva que permite configurar todos los hiperparámetros relevantes.

4.1.1. Arquitectura del Entrenador

El diseño del entrenador sigue un patrón de configuración interactiva que permite tanto a usuarios novatos como experimentados adaptar el proceso de entrenamiento a sus necesidades específicas. La función `get_hyperparameters()` implementa un sistema de configuración escalonada que presenta opciones básicas inicialmente y permite acceder a parámetros avanzados bajo demanda. Esta aproximacion equilibra la simplicidad de uso con la flexibilidad técnica necesaria para experimentación avanzada.

```python
def get_hyperparameters():
    """
    Permite al usuario configurar hiperparámetros o usar valores por defecto.
    """
    print("\n=== Configuración de Hiperparámetros del Modelo ===")
    print("Presiona Enter para usar valores por defecto, o ingresa valores personalizados:")

    hyperparams = DEFAULT_HYPERPARAMS.copy()

    # Parámetros de entrenamiento
    epochs = input(f"Épocas [{DEFAULT_HYPERPARAMS['epochs']}]: ").strip()
    if epochs:
        try:
            hyperparams["epochs"] = int(epochs)
        except ValueError:
            print("Entrada inválida, usando valor por defecto.")

    # Configuración avanzada condicional
    advanced = input("Configurar parámetros avanzados? (y/n) [n]: ").strip().lower()
    if advanced in ['y', 'yes', 's', 'si', 'sí']:
        # Configuración de early stopping
        early_stopping = input(f"Habilitar early stopping? (y/n) [y]: ").strip().lower() or "y"
        if early_stopping in ['n', 'no']:
            hyperparams["early_stopping_enabled"] = False
```

Listing 1: Configuración interactiva de hiperparámetros

El módulo maneja automáticamente la detección y configuración del hardware disponible, adaptándose tanto a sistemas con GPU como a aquellos que dependen únicamente de CPU. La configuración predeterminada de hiperparámetros ha sido optimizada empíricamente para proporcionar un balance

adecuado entre tiempo de entrenamiento y precisión del modelo, utilizando valores como 100 épocas, tamaño de lote de 16 muestras y una tasa de aprendizaje inicial de 0.001.

4.1.2. Procesamiento de Datos y Extracción de Características

La función `extract_features()` implementa el pipeline de extracción de características MFCC descrito anteriormente, procesando archivos de audio completos para generar representaciones numéricas compactas. Una ventaja fundamental del sistema es el aprovechamiento de la biblioteca Librosa, que encapsula toda la complejidad matemática del procesamiento de señales de audio y hace accesibles técnicas sofisticadas de análisis espectral que de otro modo requerirían implementaciones desde cero de algoritmos complejos como la FFT, el banco de filtros Mel y la DCT.

El sistema calcula 13 coeficientes MFCC estándar, un número que se ha establecido como óptimo en la literatura especializada para capturar la información tímbrica esencial sin introducir ruido excesivo. Los parámetros `n_fft=2048` y `hop_length=512` han sido seleccionados cuidadosamente: el tamaño de ventana FFT de 2048 muestras proporciona una resolución frecuencial adecuada para música (aproximadamente 21.5 Hz por bin con frecuencia de muestreo de 44.1 kHz), mientras que el hop length de 512 muestras asegura un solapamiento del 75 % entre ventanas consecutivas, garantizando continuidad temporal en el análisis sin incrementar excesivamente la carga computacional.

```python
def extract_features(file_path, num_mfcc=13, n_fft=2048, hop_length=512):
    """
    Extrae características MFCC de un archivo de audio completo.
    """
    try:
        # Cargar archivo de audio completo con warnings suprimidos
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            signal, sr = librosa.load(file_path, sr=SAMPLE_RATE)

        # Extraer MFCC de toda la canción
        mfcc = librosa.feature.mfcc(
            y=signal, sr=sr, n_mfcc=num_mfcc,
            n_fft=n_fft, hop_length=hop_length
        )

        # Calcular estadísticas para obtener un vector de características fijo
        mfcc_mean = np.mean(mfcc, axis=1)
        mfcc_std = np.std(mfcc, axis=1)

        # Concatenar para formar el vector de características
        features = np.concatenate((mfcc_mean, mfcc_std))
        return features

    except Exception as e:
        print(f"Error procesando {file_path}: {str(e)}")
        return None
```

Listing 2: Extracción de características MFCC de archivos completos

La estrategia de agregación estadística mediante el cálculo de medias y desviaciones estándar de los coeficientes MFCC es crucial para manejar la variabilidad temporal inherente en las piezas musicales. Esta aproximación se fundamenta en investigaciones que demuestran que las estadísticas de primer y segundo orden de los MFCC capturan eficazmente tanto las características espectrales promedio (media) como la variabilidad tímbrica (desviación estándar) de una pieza musical completa. Al reducir secuencias de coeficientes MFCC de longitud variable a un vector de dimensión fija de 26 características (13 medias + 13 desviaciones estándar), el sistema puede procesar canciones de cualquier duración manteniendo una representación consistente y computacionalmente manejable.

Para obtener una representación compacta y estable de cada archivo de audio completo, se calculan las medias y desviaciones estándar de estos coeficientes a lo largo de toda la duración de la pieza musical. Esta estrategia de agregación temporal produce un vector final de 26 características que resume eficientemente el contenido espectral de la canción completa, proporcionando una representación robusta y computacionalmente manejable para el entrenamiento del clasificador de géneros musicales.

4.1.3. Arquitectura de Red Neuronal

El modelo neuronal implementado en `build_model()` utiliza una arquitectura de red densa completamente conectada, específicamente diseñada para el procesamiento de características estadísticas extraídas del audio. La red consta de cuatro capas ocultas con 512, 256, 128 y 64 neuronas, respectivamente, cada una seguida de normalización por lotes y regularización dropout.

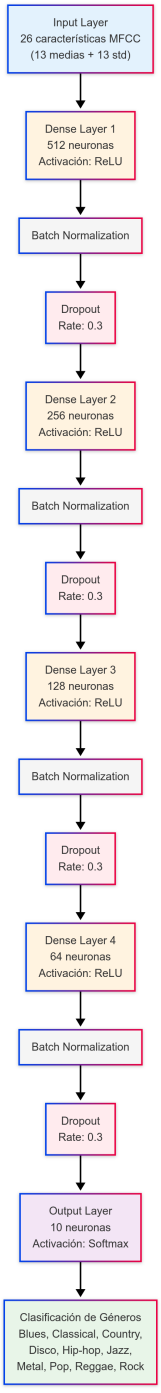

Figura 4: Arquitectura de la red neuronal densa para clasificación de géneros musicales.

```python
def build_model(input_shape, num_classes, hyperparams):
    """
    Construye el modelo para clasificación de audio.
    """
    # Crear modelo secuencial
    model = models.Sequential()

    # Capas densas para características estadísticas
    model.add(layers.Input(shape=input_shape))
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(hyperparams["dropout_rate"]))

    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(hyperparams["dropout_rate"]))

    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(hyperparams["dropout_rate"]))

    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(hyperparams["dropout_rate"]))

    model.add(layers.Dense(num_classes, activation='softmax'))

    # Compilar modelo con tasa de aprendizaje personalizada
    optimizer = tf.keras.optimizers.Adam(learning_rate=hyperparams["learning_rate"])
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

Listing 3: Construcción de la arquitectura de red neuronal

Esta configuración arquitectónica refleja un diseño conservador pero efectivo que evita tanto el sobreajuste como el subajuste. La normalización por lotes acelera el entrenamiento y mejora la estabilidad numérica, mientras que las capas dropout con tasa configurable (por defecto 0.3) proporcionan regularización efectiva. La capa de salida utiliza activación softmax para generar distribuciones de probabilidad sobre las diez clases de géneros musicales del dataset GTZAN.

4.1.4. Estrategias de Optimización y Regularización

El proceso de entrenamiento implementa múltiples estrategias de optimización que incluyen early stopping adaptativo, reducción automática de la tasa de aprendizaje y monitorización continua del rendimiento en el conjunto de validación. La función `train_model()` coordina estos mecanismos para maximizar la eficiencia del entrenamiento mientras minimiza el riesgo de sobreajuste.

```python
# Construir lista de callbacks condicionalmente
callbacks = []

# Siempre agregar ReduceLROnPlateau
callbacks.append(
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=hyperparams["reduce_lr_factor"],
        patience=hyperparams["reduce_lr_patience"],
        min_lr=hyperparams["min_lr"],
        verbose=1
    )
)

# Agregar Early Stopping solo si está habilitado
```

```
if hyperparams.get("early_stopping_enabled", True):
    callbacks.append(
        tf.keras.callbacks.EarlyStopping(
            monitor='val_accuracy',
            patience=hyperparams["early_stopping_patience"],
            restore_best_weights=True,
            verbose=1
        )
    )
```

Listing 4: Configuración de callbacks para optimización del entrenamiento

El early stopping, configurable a través de la interfaz de usuario, permite detener el entrenamiento automáticamente cuando el modelo deja de mejorar en el conjunto de validación, evitando tanto el desperdicio computacional como la degradación del rendimiento por sobreentrenamiento. La reducción programada de la tasa de aprendizaje (ReduceLROnPlateau) ajusta dinámicamente este hiperparámetro cuando la función de pérdida se estabiliza, permitiendo una convergencia más fina en las etapas finales del entrenamiento.

4.1.5. Sistema de Versionado y Persistencia

Una característica distintiva del módulo es su sistema integral de versionado y persistencia de modelos. La función `save_model()` no solo guarda el modelo entrenado y el codificador de etiquetas, sino que también preserva todos los metadatos relevantes, incluyendo hiperparámetros utilizados, precisión final alcanzada, fecha de entrenamiento y parámetros de preprocesamiento.

```
def save_model(model, encoder, hyperparams, model_name=None, test_accuracy=None):
    """
    Guarda el modelo entrenado y codificador de etiquetas con soporte de versionado.
    """
    # Generar nombre del modelo si no se proporciona
    if model_name is None:
        timestamp = time.strftime("%Y%m%d_%H%M%S")
        model_name = f"model_{timestamp}"

    # Crear directorio específico del modelo
    model_dir = f"models/{model_name}"
    os.makedirs(model_dir, exist_ok=True)

    # Guardar información del modelo como JSON para fácil lectura
    info = {
        "model_name": model_name,
        "training_date": time.strftime("%Y-%m-%d %H:%M:%S"),
        "genres": GENRES,
        "sample_rate": SAMPLE_RATE,
        "architecture": "Dense Neural Network",
        "input_features": "MFCC statistics (mean + std)",
        "hyperparameters": hyperparams,
        "test_accuracy": round(test_accuracy, 4) if test_accuracy else None
    }

    info_path = f"{model_dir}/model_info.json"
    with open(info_path, "w") as f:
        json.dump(info, f, indent=2)
```

Listing 5: Sistema de persistencia y metadatos del modelo

Este sistema facilita la reproducibilidad de experimentos y permite la gestión eficiente de múltiples versiones de modelos. Cada modelo entrenado se almacena en un directorio independiente que incluye visualizaciones automáticamente generadas del proceso de entrenamiento, matrices de confusión detalladas y archivos JSON con información legible sobre la configuración del modelo. Esta aproximación integral al almacenamiento de modelos facilita tanto la evaluación comparativa como la selección informada de modelos para producción.

4.2. Módulo Clasificador de Audio

El módulo clasificador de audio, implementado en `audio_classifier.py`, constituye el núcleo funcional del sistema que proporciona las capacidades de clasificación de géneros musicales a los demás componentes. Este módulo está diseñado como una biblioteca de funciones que pueden ser utilizadas tanto de forma independiente como integradas en aplicaciones más complejas, ofreciendo una interfaz limpia y consistente para la carga de modelos y clasificación de archivos de audio.

4.2.1. Gestión y Carga de Modelos

El sistema implementa un mecanismo robusto de gestión de modelos que permite la detección automática y carga de cualquier modelo previamente entrenado. La función `load_model_and_params()` coordina la carga completa del ecosistema de archivos necesarios para la clasificación, incluyendo el modelo neuronal, el codificador de etiquetas y los parámetros de preprocesamiento.

```python
def load_model_and_params(model_name=None):
    """
    Carga el modelo entrenado y parámetros de preprocesamiento.
    """
    if model_name is None:
        model_name = select_model()
        if model_name is None:
            return None, None, None

    model_dir = f"models/{model_name}"

    # Verificar existencia de archivos necesarios
    model_path = f"{model_dir}/music_genre_classifier.h5"
    encoder_path = f"{model_dir}/label_encoder.pkl"
    params_path = f"{model_dir}/preprocessing_params.pkl"

    try:
        # Cargar modelo
        model = load_model(model_path)

        # Cargar codificador
        with open(encoder_path, "rb") as f:
            encoder = pickle.load(f)

        # Cargar parámetros
        with open(params_path, "rb") as f:
            params = pickle.load(f)

        return model, encoder, params

    except Exception as e:
        print(f"Error cargando modelo: {str(e)}")
        return None, None, None
```

Listing 6: Carga integral de modelos y parámetros

Esta aproximación garantiza la compatibilidad completa entre todos los componentes del sistema y permite la coexistencia de múltiples versiones de modelos sin conflictos, facilitando tanto la experimentación como el despliegue en producción.

4.2.2. Pipeline de Clasificación

La función central `classify_audio()` implementa el pipeline completo de clasificación, desde la extracción de características hasta la generación de predicciones probabilísticas. El sistema reutiliza exactamente la misma metodología de extracción de características empleada durante el entrenamiento, garantizando consistencia en el preprocesamiento de datos y eliminando posibles discrepancias que

podrían degradar el rendimiento del modelo.

```python
def classify_audio(file_path, model, encoder, params):
    """
    Clasifica un archivo de audio en uno de los géneros musicales.
    """
    # Extraer características
    features = extract_features(file_path, params)

    if features is None:
        return None, None, None

    # Preparar para predicción
    X = np.array([features])

    # Predecir con medición de tiempo
    start_time = time.time()
    predictions = model.predict(X)
    prediction_time = time.time() - start_time

    # Obtener probabilidades
    probabilities = predictions[0]

    # Obtener género con mayor probabilidad
    genre_idx = np.argmax(probabilities)
    genre = params["genres"][genre_idx]

    return genre, probabilities, prediction_time
```

Listing 7: Pipeline de clasificación de audio con medición de rendimiento

El sistema proporciona no solo la clasificación final sino también las probabilidades asociadas a cada género y métricas de rendimiento temporal, permitiendo a las aplicaciones cliente evaluar la confianza de las predicciones y optimizar los flujos de trabajo según los requisitos de latencia.

4.2.3. Capacidades de Visualización

Aunque secundarias a la funcionalidad principal de clasificación, el módulo incluye herramientas de visualización que facilitan la interpretación de resultados y el análisis del contenido espectral. La función `visualize_prediction()` genera representaciones gráficas de las distribuciones de probabilidad, mientras que `visualize_audio()` proporciona análisis espectrales detallados del contenido de audio.

```python
def visualize_prediction(probabilities, genres):
    """
    Visualiza las probabilidades de predicción.
    """
    # Ordenar géneros por probabilidad para mejor visualización
    sorted_indices = np.argsort(probabilities)[::-1]
    sorted_genres = [genres[i] for i in sorted_indices]
    sorted_probs = probabilities[sorted_indices]

    plt.figure(figsize=(12, 6))
    bars = plt.bar(sorted_genres, sorted_probs * 100)

    # Colorear la barra del género con mayor probabilidad
    bars[0].set_color('green')

    plt.xlabel('Género Musical')
    plt.ylabel('Probabilidad (%)')
    plt.title('Predicción de Género Musical')
    plt.xticks(rotation=45)
```

Listing 8: Visualización de predicciones ordenadas por probabilidad

Estas herramientas de visualización son especialmente útiles durante el desarrollo y depuración de aplicaciones que utilizan el módulo, así como para presentaciones y demostraciones del sistema en funcionamiento.

4.3. Módulo de Generación de Contenido Visual

El módulo de generación de contenido visual, implementado en `text_to_image_generator.py`, constituye el componente final del pipeline de transformación, donde las características musicales extraídas se convierten en imágenes y videos mediante modelos de difusión. Este módulo integra las funcionalidades de análisis de audio, generación de prompts descriptivos y síntesis visual, proporcionando una solución completa para la conversión de contenido musical en representaciones visuales.

4.3.1. Configuración y Gestión de Modelos de Difusión

La función `setup_model()` implementa un sistema robusto de detección y configuración de hardware que optimiza automáticamente los parámetros de los modelos según los recursos disponibles. El sistema utiliza el modelo *dreamlike-art/dreamlike-diffusion-1.0* para generación de imágenes y *stabilityai/stable-video-diffusion-img2vid-xt* para animación de videos, ambos modelos de código abierto que garantizan el acceso sin costos de licencia.

```
def setup_model(model_type="image"):
    """
    Configura y carga el modelo Stable Diffusion.
    """
    model_id = "dreamlike-art/dreamlike-diffusion-1.0"

    print(f"Cargando modelo {model_type}: {model_id}")

    # Configurar PyTorch para mejor gestión de memoria
    torch.cuda.empty_cache()

    # Verificar disponibilidad CUDA
    if torch.cuda.is_available():
        device = "cuda"
        print("Usando GPU (CUDA)")

        # Mostrar información de memoria GPU
        gpu_mem_total = torch.cuda.get_device_properties(0).total_memory / 1024**3
        gpu_mem_reserved = torch.cuda.memory_reserved(0) / 1024**3
        gpu_mem_allocated = torch.cuda.memory_allocated(0) / 1024**3
        print(f"Memoria GPU total: {gpu_mem_total:.2f} GB")
        print(f"Memoria GPU reservada: {gpu_mem_reserved:.2f} GB")
        print(f"Memoria GPU asignada: {gpu_mem_allocated:.2f} GB")

        is_cpu = False
    else:
        device = "cpu"
        print("\nADVERTENCIA! CUDA no disponible, usando CPU.")
        print("La generación por CPU puede tardar MUCHO tiempo (10-30 minutos por imagen/video).")
        print("Se recomienda una GPU con CUDA para este programa.")
        is_cpu = True
```

Listing 9: Configuración adaptativa de modelos de difusión

El sistema implementa optimizaciones específicas para cada modalidad de hardware, incluyendo el uso de precisión half-float (FP16) en GPU para reducir el consumo de memoria, y configuraciones especiales de offloading y attention slicing para maximizar la eficiencia en sistemas con recursos limitados.

4.3.2. Análisis Avanzado de Características Musicales

Una contribución significativa del sistema es la implementación de un analizador exhaustivo de características musicales en la función `extract_song_features_enhanced()`. Este componente realiza un análisis multidimensional del contenido de audio que va más allá de la simple clasificación de géneros, extrayendo información detallada sobre instrumentación, características emocionales y estructura musical.

```
# === DETECCIÓN DE INSTRUMENTOS ===
# Análisis de bandas de frecuencia para instrumentos comunes
instruments = []

# Extraer características espectrales
freq_bands = librosa.fft_frequencies(sr=sr)

# Bandas de frecuencia aproximadas para varios instrumentos
instrument_bands = {
    'bass': (40, 250),
    'kick drum': (50, 100),
    'electric guitar': (300, 4000),
    'piano': (27, 4200),
    'synthesizer': (80, 8000),
    'strings': (200, 3500),
    'trumpet': (160, 980),
    'saxophone': (100, 900),
    'drums': (200, 12000),
}

# Detectar presencia de instrumentos por energía en sus bandas típicas
for instrument, (low_freq, high_freq) in instrument_bands.items():
    band_energy = np.mean(spec[(freq_bands >= low_freq) & (freq_bands <= high_freq)])
    instrument_energy = band_energy / np.mean(spec)  # Normalizar
    if instrument_energy > 0.6:
        instruments.append(instrument)
```

Listing 10: Detección automática de instrumentos mediante análisis espectral

El análisis de instrumentación utiliza un enfoque basado en la distribución energética en bandas de frecuencia características, permitiendo la identificación automática de instrumentos predominantes sin requerir modelos de aprendizaje automático especializados.

4.3.3. Análisis Emocional y Mapeo Afectivo

Para el análisis emocional, el sistema implementa un modelo computacional basado en el circumplex de Russell que mapea características acústicas a dimensiones emocionales de valencia (positivo/negativo) y activación (alta/baja energía). Esta aproximación, aunque no utiliza modelos de inteligencia artificial dedicados al análisis de sentimientos, ha demostrado ser efectiva y constituye una base sólida que puede mejorarse en futuras iteraciones del sistema.

```
# === ANÁLISIS EMOCIONAL DETALLADO ===
# Modelo básico de emociones musicales basado en características acústicas

# Extraer características adicionales para análisis emocional
chroma = np.mean(librosa.feature.chroma_stft(y=y, sr=sr), axis=1)
spectral_contrast = np.mean(librosa.feature.spectral_contrast(y=y, sr=sr), axis=1)

# Calcular características para mapeo emocional
tempo_norm = (tempo - 60) / 120  # Normalizar tempo (60-180 BPM)

# Modos mayor/menor (aproximación)
major_minor_ratio = np.std(chroma) / np.mean(chroma)
is_major = major_minor_ratio < 0.15
```

```
# Presencia de disonancia
dissonance = np.mean(spectral_contrast[1:3])

# Calcular valores emocionales
emotions = {}

# Valencia (positivo vs negativo)
emotions['valence'] = 0.6 if is_major else 0.3  # Basado en modo mayor/menor
emotions['valence'] += 0.2 * energy_percentile  # Energía alta  más positivo
emotions['valence'] -= 0.1 * dissonance  # Disonancia  más negativo
emotions['valence'] = max(0, min(1, emotions['valence']))  # Limitar 0-1

# Activación (intensidad)
emotions['arousal'] = 0.3 * energy_percentile + 0.3 * tempo_norm + 0.2 * dissonance
emotions['arousal'] = max(0, min(1, emotions['arousal']))  # Limitar 0-1
```

Listing 11: Mapeo computacional de características acústicas a emociones

La decisión de implementar un análisis emocional basado en reglas computacionales en lugar de modelos de IA especializados responde a consideraciones de fiabilidad y control. Los modelos gratuitos disponibles para análisis de sentimientos musicales mostraron inconsistencias significativas durante las pruebas preliminares, mientras que el enfoque implementado proporciona resultados predecibles y coherentes que pueden ser refinados sistemáticamente.

4.3.4. Generación Inteligente de Prompts

La función `song_to_prompt()` representa el núcleo conceptual del sistema, donde toda la información extraída del análisis musical se transforma en descripciones textuales optimizadas para modelos de difusión. Este proceso combina las descripciones base por género con las características específicas detectadas en cada canción individual.

```
def song_to_prompt(file_path, genre):
    """
    Genera un prompt descriptivo para una canción específica, considerando
    tanto el género como las características específicas del audio.
    """
    # Extraer características específicas de la canción
    features = extract_song_features_enhanced(file_path)

    if features is None:
        print("No se pudieron extraer características específicas. Usando solo el género.")
        return music_genre_to_prompt_static(genre)

    # Usar el método estático como base para asegurar un resultado útil
    base_prompt = music_genre_to_prompt_static(genre)

    # Construir descripción enriquecida basada en las características extraídas
    instruments_str = ", ".join(features.get("instruments", [])) if features.get("instruments")
↪ else "instrumentos típicos"
    emotions_str = ", ".join(features.get("emotions", [])) if features.get("emotions") else
↪ "emociones características"

    # Construir prompt enriquecido
    enhanced_prompt = f"{base_prompt}, con tempo {features['tempo_category']}
↪ ({features['tempo']} BPM), energía alta"

    # Añadir detalles de instrumentos detectados
    if features.get("instruments"):
        enhanced_prompt += f", destacando {instruments_str}"

    # Añadir descripción de emociones detectadas
    if features.get("emotions"):
```

```
        enhanced_prompt += f", atmósfera {emotions_str}"

    # Añadir información sobre voz
    if features.get("has_vocals", False):
        enhanced_prompt += f", con voz {features.get('vocal_clarity', 'prominente')}"
    else:
        enhanced_prompt += ", composición instrumental"

    return enhanced_prompt
```

Listing 12: Construcción de prompts enriquecidos con análisis musical

Esta integración permite generar imágenes que no solo reflejan el género musical general, sino que capturan las características específicas y el carácter emocional de cada pieza individual, resultando en representaciones visuales más precisas y personalizadas.

4.3.5. Pipeline de Generación Visual

Las funciones `generate_image()` y `generate_video()` implementan el pipeline completo de síntesis visual, incluyendo gestión automática de memoria, optimización de parámetros según el hardware disponible, y manejo robusto de errores. El sistema proporciona retroalimentación detallada sobre el progreso de generación y métricas de rendimiento para facilitar la optimización de flujos de trabajo.

```
def generate_image(pipe, prompt, params=None, is_cpu=False):
    """
    Genera imágenes desde un prompt usando el pipeline Stable Diffusion.
    """
    if is_cpu:
        print("\nGenerando imagen en CPU. Esto puede tardar mucho tiempo...")
        print("Por favor espera pacientemente. No cierres el programa.")

        # Mostrar contador de tiempo
        import time
        start_time = time.time()

        def callback_fn(step, timestep, latents):
            elapsed = time.time() - start_time
            print(f"Paso {step}/{params.get('num_inference_steps', 50)} - Tiempo transcurrido:
  ↪ {elapsed:.1f} segundos", end="\r")
            return None

        # Añadir callback para mostrar progreso
        params['callback'] = callback_fn

    # Generar imágenes
    images = pipe(prompt, **params).images

    if is_cpu:
        total_time = time.time() - start_time
        print(f"\nGeneración completada en {total_time:.1f} segundos")

    return images
```

Listing 13: Generación de imágenes con gestión automática de recursos

El módulo incluye funcionalidades adicionales para persistencia de resultados, con opciones de guardado tanto para imágenes estáticas como para videos en múltiples formatos (PNG, GIF, MP4), facilitando la integración con flujos de trabajo posteriores y la distribución de contenido generado.

4.4. Aplicación Web Streamlit

La aplicación web, implementada en `app_streamlit.py`, constituye la interfaz de usuario principal del sistema Sound to Something, integrando todos los módulos previamente descritos en una experiencia

de usuario coherente y accesible. Esta aplicación web permite a los usuarios interactuar con el sistema completo sin necesidad de conocimientos técnicos obligatoriamente, proporcionando un flujo de trabajo guiado desde la carga de audio hasta la generación de contenido visual.

4.4.1. Arquitectura de la Interfaz Web

La aplicación utiliza Streamlit como framework de desarrollo web, aprovechando sus capacidades para crear interfaces interactivas con mínimo código HTML/CSS personalizado. El sistema implementa un diseño basado en estados que guía al usuario através de cinco pasos principales: bienvenida, carga de audio, análisis de resultados, selección de tipo de generación, y visualización de resultados finales.

```
# Configuración de la página
st.set_page_config(
    page_title="Sound to Something",
    page_icon=(Cualquier icono, aunque no puedo ponerlo porque LaTeX no permite mostrarlos aquí.),
    layout="wide",
    initial_sidebar_state="collapsed"
)

# Inicializar el estado de la sesión si no existe
if 'step' not in st.session_state:
    st.session_state.step = 'welcome'

if 'audio_file' not in st.session_state:
    st.session_state.audio_file = None

if 'generated_images' not in st.session_state:
    st.session_state.generated_images = []

# Funciones de navegación
def go_to_step(step):
    st.session_state.step = step

def start_over():
    st.session_state.step = 'upload'
    st.session_state.audio_file = None
    st.session_state.spectrogram = None
    # Reset all session state variables
```

Listing 14: Configuración inicial y gestión de estados de la aplicación

La gestión de estados permite mantener la coherencia de datos entre las diferentes pantallas de la aplicación, asegurando que la información extraída del audio y los parámetros de generación se preserven durante toda la sesión del usuario.

4.4.2. Sistema de Gestión de Modelos Dinámico

Una característica destacada de la aplicación es su sistema de gestión de modelos que permite la selección dinámica entre diferentes versiones de clasificadores entrenados. La función `setup_sidebar()` implementa una barra lateral que detecta automáticamente los modelos disponibles y presenta información detallada sobre cada uno, incluyendo fecha de entrenamiento, número de épocas y precisión alcanzada.

```
def setup_sidebar():
    """
    Configura la barra lateral con opciones de selección de modelo.
    """
    st.sidebar.markdown("## Model Configuration")

    # Obtener modelos disponibles
    available_models = list_available_models()
```

```
    if not available_models:
        st.sidebar.warning("No trained models found")
        return None

    # Mostrar información de modelos disponibles
    model_info = {}
    for model in available_models:
        try:
            info_path = f"models/{model}/model_info.json"
            if os.path.exists(info_path):
                with open(info_path, "r") as f:
                    info = json.load(f)
                training_date = info.get('training_date', 'Unknown')
                accuracy = info.get('test_accuracy', 'N/A')

                st.sidebar.markdown(f"""
                **{model}**
                - Trained: {training_date}
                - Accuracy: {accuracy if accuracy != 'N/A' else 'N/A'}
                """)
```

Listing 15: Detección y selección automática de modelos entrenados

Este sistema permite a los usuarios experimentar con diferentes versiones de modelos y comparar su rendimiento sin necesidad de modificar código o archivos de configuración.

4.4.3. Pipeline de Análisis Integrado

La función `analyze_audio()` orquesta la ejecución completa del pipeline de análisis, coordinando la carga de modelos, la extracción de características, la clasificación de géneros y la generación de prompts. Esta función implementa manejo robusto de errores y proporciona retroalimentación detallada al usuario sobre el progreso de cada etapa.

```
def analyze_audio(audio_file):
    if audio_file is None:
        return None, None, None, None, "Please upload an audio file"

    st.info("Starting audio analysis...")

    # Get selected model from session state
    selected_model = st.session_state.get('selected_model', None)

    # Load models with selected configuration
    models_data = load_models(selected_model)
    if models_data is None:
        return None, None, None, None, "Failed to load models"

    # Get classifier components
    classifier_model, encoder, params = models_data["classifier"]

    # Save uploaded audio to temporary file
    file_path = save_uploaded_file(audio_file)

    # Generate visualizations
    spectrogram_path = visualize_audio(file_path)

    # Classify genre
    genre, probabilities, prediction_time = classify_audio(
        file_path, classifier_model, encoder, params
    )

    # Extract enhanced features
    features = extract_song_features_enhanced(file_path)
```

```
    # Generate optimized prompt
    prompt = song_to_prompt(file_path, genre)

    return file_path, spectrogram_path, probs_chart_path, prompt, features_html
```

Listing 16: Coordinación del pipeline completo de análisis de audio

La función coordina automáticamente todas las etapas del análisis y genera tanto visualizaciones técnicas (espectrogramas, distribuciones de probabilidad) como descripciones textuales optimizadas para la generación visual posterior.

4.4.4. Generación Visual Adaptativa

Las funciones de generación visual `generate_image_from_prompt()` y `generate_video_from_image()` implementan interfaces adaptativas que ajustan automáticamente los parámetros de generación según las capacidades del hardware detectado. El sistema proporciona configuración granular de parámetros avanzados mientras mantiene valores predeterminados optimizados para usuarios no técnicos.

```
def generate_image_from_prompt(prompt, steps, width, height, num_images,
                               negative_prompt="", guidance_scale=7.5, seed=-1):
    # Load models with hardware detection
    models_data = load_models()
    if models_data is None:
        return None, "Failed to load models"

    # Get image generator and hardware configuration
    image_pipe = models_data["image_generator"]
    is_cpu = models_data["is_cpu"]

    # Configure generation parameters
    image_params = {
        'num_inference_steps': steps,
        'width': width,
        'height': height,
        'num_images_per_prompt': num_images,
        'guidance_scale': guidance_scale
    }

    # Add hardware-specific optimizations
    if seed != -1:
        image_params["generator"] = torch.Generator().manual_seed(seed)

    if negative_prompt:
        image_params['negative_prompt'] = negative_prompt

    # Generate with progress feedback
    with st.spinner("Generating images..."):
        images = generate_image(image_pipe, prompt, image_params, is_cpu)

    return image_data, None
```

Listing 17: Generación de imágenes con configuración adaptativa de hardware

También implementa optimizaciones específicas para cada modalidad de hardware, incluyendo ajustes automáticos de memoria, configuración de precisión numérica y gestión de recursos computacionales.

4.4.5. Interfaz de Usuario

La aplicación implementa un diseño responsive que se adapta dinámicamente al contenido generado y al progreso del usuario. El sistema utiliza un esquema de navegación por pasos que permite tanto el avance lineal como el retroceso para modificar parámetros, facilitando la experimentación iterativa

con diferentes configuraciones.

```
# PASO 3: Resultados del análisis
elif st.session_state.step == 'analysis':
    st.markdown('<h2 class="section-title">Step 2: Audio Analysis Results</h2>',
                unsafe_allow_html=True)

    col1, col2 = st.columns(2)

    with col1:
        st.markdown('<h3 class="subsection-title">Waveform and Spectrogram</h3>',
                    unsafe_allow_html=True)
        if st.session_state.spectrogram and os.path.exists(st.session_state.spectrogram):
            st.image(st.session_state.spectrogram)

        st.markdown('<h3 class="subsection-title">Genre Classification</h3>',
                    unsafe_allow_html=True)
        if st.session_state.genre_probs and os.path.exists(st.session_state.genre_probs):
            st.image(st.session_state.genre_probs)

    with col2:
        st.markdown('<h3 class="subsection-title">Detailed Audio Features</h3>',
                    unsafe_allow_html=True)
        if st.session_state.features_html:
            st.markdown(st.session_state.features_html, unsafe_allow_html=True)

        # Editable prompt generation
        prompt = st.text_area("You can edit this prompt if you want:",
                              value=st.session_state.prompt, height=100)
        st.session_state.prompt = prompt
```

Listing 18: Gestión dinámica de la interfaz según el estado de la aplicación

La interfaz presenta información compleja de manera organizada y proporciona controles intuitivos para la edición de parámetros, permitiendo a los usuarios personalizar el proceso de generación según sus preferencias artísticas.

4.4.6. Gestión de Recursos y Limpieza Automática

La aplicación implementa un sistema automático de gestión de archivos temporales que previene la acumulación excesiva de datos en el sistema de archivos. El sistema crea, utiliza y elimina automáticamente archivos de trabajo manteniendo un equilibrio entre la funcionalidad requerida y el uso eficiente del almacenamiento.

```
# Clean temp files
try:
    temp_files = os.listdir("temp")
    current_time = time.time()
    for file in temp_files:
        file_path = os.path.join("temp", file)
        # Delete files older than 1 hour
        if os.path.isfile(file_path) and current_time - os.path.getmtime(file_path) > 3600:
            os.remove(file_path)
except Exception as e:
    print(f"Error cleaning temp files: {e}")

# Function to save uploaded file temporarily
def save_uploaded_file(uploaded_file):
    try:
        temp_dir = "temp"
        os.makedirs(temp_dir, exist_ok=True)

        file_path = os.path.join(temp_dir, uploaded_file.name)
```

```
        # Save the uploaded file
        with open(file_path, "wb") as f:
            f.write(uploaded_file.getbuffer())

        return file_path
    except Exception as e:
        st.error(f"Error saving file: {e}")
        return None
```

Listing 19: Sistema de limpieza automática de archivos temporales

Este enfoque asegura que la aplicación pueda funcionar continuamente sin degradación del rendimiento por acumulación de archivos temporales, mientras mantiene los datos necesarios durante la sesión activa del usuario.

4.4.7. Ejecución de la Aplicación

La aplicación web integra de manera transparente todos los módulos del sistema, proporcionando acceso completo a las capacidades de clasificación musical, análisis de características y generación visual a través de una interfaz web intuitiva. El sistema está diseñado para ejecutarse localmente, aprovechando los recursos de hardware disponibles mientras proporciona una experiencia de usuario similar a aplicaciones web modernas.

Para ejecutar la aplicación completa, se utiliza el siguiente comando en la terminal, desde el directorio que contiene el archivo `app_streamlit.py`:

```
streamlit run app_streamlit.py --server.fileWatcherType none
```

Listing 20: Comando de ejecución de la aplicación web

5. Evaluación

5.1. Verificación del Sistema Completo

Para una evaluación completa del funcionamiento del sistema Sound to Something, se recomienda la descarga e instalación del proyecto completo, el cual se encuentra adjunto como material complementario de esta memoria. Esta aproximación permite verificar todas las funcionalidades implementadas y experimentar con diferentes tipos de contenido musical, proporcionando una comprensión práctica de las capacidades y limitaciones del sistema desarrollado.

5.1.1. Proceso de Instalación y Configuración

El sistema requiere la instalación de múltiples dependencias especializadas que se encuentran detalladas en el archivo `requirements.txt` incluido en el proyecto. Para una experiencia óptima, se recomienda encarecidamente la instalación de las librerías con soporte CUDA, lo que permite aprovechar las capacidades de aceleración por GPU para los modelos de difusión. Aunque el proceso de configuración de CUDA puede ser extenso y específico para cada sistema operativo y versión de controladores, existen múltiples guías especializadas disponibles que detallan este procedimiento paso a paso.

a instalación básica del entorno en Python, junto con sus dependencias, puede realizarse utilizando los comandos estándar de gestión de paquetes. No obstante, también es posible emplear otras alternativas, como Anaconda, según las preferencias del usuario o los requerimientos del sistema:

```
# Crear entorno virtual (recomendado)
python -m venv sound_to_something_env
```

```
# Activar entorno virtual
# En Windows:
sound_to_something_env\Scripts\activate
# En Linux/Mac:
source sound_to_something_env/bin/activate

# Instalar dependencias
pip install -r requirements.txt

# Verificar instalación de PyTorch con CUDA (opcional pero recomendado)
python -c "import torch; print(f'CUDA disponible: {torch.cuda.is_available()}')"
```

Listing 21: Instalación de dependencias del proyecto

Es importante verificar que todas las dependencias se instalen correctamente, prestando especial atención a las versiones específicas de PyTorch y TensorFlow que garantizan la compatibilidad entre los diferentes componentes del sistema.

5.1.2. Visualización

A continuación se muestran dos ejemplos del sistema en funcionamiento: la pantalla inicial de la aplicación web y el resultado obtenido tras clasificar y procesar un audio del género rock.

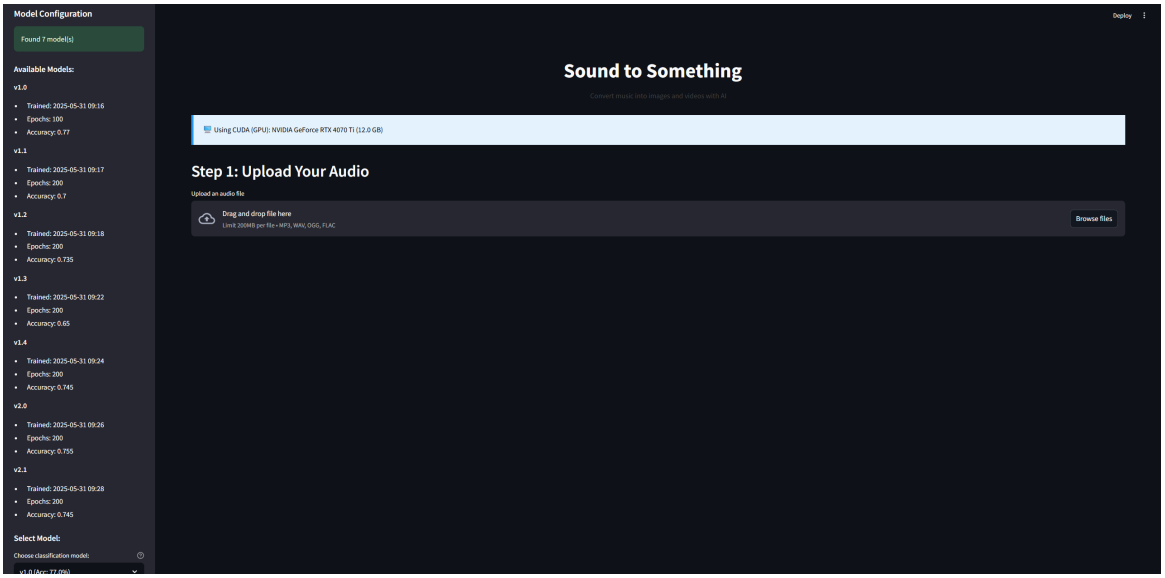

Figura 5: Pantalla principal de la aplicación web Sound to Something.

Figura 6: Imagen generada automáticamente tras clasificar un audio como rock.

Estos ejemplos ilustran tanto la interfaz de usuario como la calidad general de los resultados obtenidos a través del pipeline completo. Es importante tener en cuenta que, dado que el sistema se ha desarrollado utilizando modelos preentrenados de acceso gratuito y ejecutados localmente en un entorno personal, la precisión de los resultados puede estar limitada. No obstante, estos ejemplos son suficientes para transmitir claramente el funcionamiento del sistema y la coherencia general entre las etapas de análisis y generación.

El proceso detallado de análisis y generación puede explorarse de forma individual siguiendo las instrucciones de instalación proporcionadas, lo que permite adquirir una comprensión práctica y directa del funcionamiento del sistema. Por este motivo, no se profundizará más en esta sección.

Igualmente, he creado un video para que se pueda ver todo el proceso completo:
https://drive.google.com/file/d/1YP-WJA95RqhjUtdijacLl29EsO40G8Bl/view?usp=drive_link

5.1.3. Análisis de la Calidad de los Resultados

El ejemplo presentado demuestra la capacidad del sistema para establecer conexiones coherentes entre las características acústicas extraídas del audio y las representaciones visuales generadas. En este caso, el análisis automático de género identificó la pieza como perteneciente al género rock, aunque es importante señalar que esta clasificación puede estar sujeta a ambigüedad, dado el solapamiento estilístico frecuente entre géneros como el rock y el metal. Esta proximidad entre estilos implica que, incluso en caso de cierta imprecisión, la salida generada sigue siendo razonablemente coherente en

términos estéticos y contextuales.

Posteriormente, el sistema de generación de prompts combinó esta clasificación con otros aspectos extraídos del audio, como la instrumentación predominante, el tempo y las características emocionales detectadas. La imagen resultante refleja adecuadamente los rasgos visuales esperados para una pieza de este estilo, incluyendo una paleta de colores intensa, composiciones dramáticas y elementos simbólicos asociados a la estética del rock.

Esta coherencia entre el análisis musical y la representación visual valida la efectividad del pipeline actual. Sin embargo, cabe destacar que una mejora significativa en la precisión y la riqueza del análisis requeriría un conjunto de datos mucho más amplio y diverso, idealmente comparable al volumen y variedad de plataformas como Spotify, que incluyen múltiples géneros, subgéneros y contextos culturales.

Referencias

[Davis and Mermelstein, 1980] Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366.

[Li et al., 2003] Li, T., Ogihara, M., and Li, Q. (2003). A comparative study on content-based music genre classification. *Proceedings of the 26th annual international ACM SIGIR conference*, pages 282–289.

[Mermelstein, 1976] Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, pages 374–388.

[Rabiner and Juang, 1993] Rabiner, L. and Juang, B.-H. (1993). *Fundamentals of speech recognition*. Prentice hall.

[Sakar and Kursun, 2013] Sakar, C. O. and Kursun, O. (2013). Detecting patients with parkinson's disease using mel frequency cepstral coefficients and support vector machines. *International Conference on Signal Processing and Communication Application (SIU)*, pages 1–4.

[Stevens et al., 1937] Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude of pitch. *The journal of the acoustical society of America*, 8(3):185–190.

[Tzanetakis and Cook, 2002] Tzanetakis, G. and Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302.