Implementación de una Red Neuronal Convolucional para la clasificación de audios mediante espectrogramas

Alexis Horacio López Fragoso

Universidad Autónoma de México,
Facultad de Estudios Superiores Cuautitlán,
Cuautitlán, Estado de México, México
guados123@gmail.com

Abstract. En este proyecto se presenta la implementación de una Red Neuronal Convolucional para la clasificación de audios mediante espectrogramas. Se utiliza Python junto con la librería librosa para el procesamiento de los audios y su conversión a espectrogramas. Para el entrenamiento de la CNN, se emplea el dataset de Audio MNIST [1]. Además para el entrenamiento de la red neuronal Convolucional se utilizara Python con las librerías Keras y scikit-learn, para así lograr la clasificación de los audios. Además, se integra un Arduino Uno con una pantalla OLED de 128x64 para la visualización de los datos de salida, utilizando la CNN previamente entrenada. Los resultados nos mostrarán la eficacia del modelo en la clasificación de patrones de audio, ofreciendo una solución viable para aplicaciones en reconocimiento de audio y procesamiento de señales de audio.

Keywords: Red Neuronal Convolucional, clasificaci'on de audio, espectrogramas, Python, librosa, Keras, sklearn, Arduino Uno, OLED 128x64

1 Introducción

El reconocimiento y clasificación de señales de audio es un campo de investigación que ha experimentado un crecimiento significativo en los últimos años, gracias al avance de las técnicas de aprendizaje profundo [2]. Las Redes Neuronales Convolucionales (CNNs) han demostrado ser especialmente eficaces en tareas de clasificación de imágenes y análisis de espectrogramas [3].
La transformación de señales de audio en representaciones visuales como espectrogramas permite aprovechar las capacidades de las CNNs en el procesamiento de audio [4]. Para este propósito, se utiliza la librería librosa en Python, que facilita el procesamiento y análisis de señales de audio [5].
En este proyecto, se propone la implementación de una CNN para la clasificación de audios utilizando el conjunto de datos Audio MNIST [6], el cual contiene muestras de voz pronunciando los dígitos del 0 al 9. El entrenamiento del modelo se llevará a cabo empleando Keras y sklearn, herramientas ampliamente

utilizadas en el aprendizaje automático [7, 8].
Además, se integrará un Arduino Uno con una pantalla OLED de 128x64 para la visualización de los resultados obtenidos por la red neuronal. Esta implementación permitirá demostrar la viabilidad de utilizar sistemas embebidos para tareas de clasificación de audio en tiempo real [9].
El objetivo principal es demostrar la eficacia de las CNNs en la clasificación de señales de audio y su aplicabilidad en dispositivos de bajo costo, contribuyendo así al desarrollo de soluciones accesibles en el área de reconocimiento de patrones y procesamiento de señales.

2 Estado del Arte

El reconocimiento y clasificación de señales de audio ha experimentado un avance significativo con la aplicación de técnicas de aprendizaje profundo [2]. A continuación, se presentan las principales contribuciones en este campo, enfocándose en el uso de espectrogramas, redes neuronales convolucionales y su implementación en sistemas embebidos.

2.1 Aprendizaje Profundo en Audio

El aprendizaje profundo ha revolucionado el procesamiento de señales de audio, permitiendo modelos que capturan características complejas de las señales [3]. Redes neuronales profundas han demostrado ser efectivas en tareas como reconocimiento de voz y clasificación de eventos sonoros [10].

2.2 Espectrogramas como Representación de Audio

La transformada de Fourier de corto tiempo (STFT) es una herramienta fundamental para convertir señales de audio en espectrogramas, proporcionando una representación tiempo-frecuencia [11]. Esta representación es adecuada para ser procesada por CNNs, ya que permite aplicar técnicas de visión por computador al dominio del audio [4].

2.3 Redes Neuronales Convolucionales en Audio

Las CNNs han sido ampliamente utilizadas en clasificación de imágenes y han mostrado un rendimiento sobresaliente en clasificación de espectrogramas [12]. Estudios previos han demostrado que las CNNs pueden aprender características discriminativas de los espectrogramas para tareas de clasificación de audio [13].

2.4 Datasets para Entrenamiento de Modelos de Audio

La disponibilidad de datasets es crucial para entrenar y evaluar modelos de aprendizaje profundo. El dataset Audio MNIST [6] es ampliamente utilizado para tareas de reconocimiento de dígitos hablados, proporcionando una base estandarizada para comparación de modelos.

2.5 Implementación en Sistemas Embebidos

La implementación de modelos de aprendizaje profundo en sistemas embebidos permite el despliegue de aplicaciones en dispositivos de bajo costo y recursos limitados [14]. El uso de microcontroladores como Arduino Uno y pantallas OLED facilita la interacción y visualización de resultados en tiempo real [9].

3 Conocimientos Previos

Para comprender la implementación de una Red Neuronal Convolucional (CNN) para la clasificación de audios mediante espectrogramas, es esencial tener una base teórica en procesamiento de señales, análisis de espectrogramas y aprendizaje profundo. A continuación, se presentan los conceptos clave necesarios.

3.1 Procesamiento de Señales de Audio

Las señales de audio son representaciones analógicas o digitales de ondas sonoras que varían en amplitud con respecto al tiempo [15]. En el dominio temporal, una señal discreta de audio se representa como una sucesión de muestras $x[n]$, donde n es el índice de tiempo discreto. El análisis y procesamiento de estas señales requieren transformaciones que permitan examinar sus componentes frecuenciales.

3.2 Transformada de Fourier y Espectrogramas

La Transformada de Fourier Discreta (DFT) es una herramienta fundamental para convertir una señal en el dominio temporal a una representación en el dominio frecuencial [15]. La DFT de una señal $x[n]$ está dada por:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn} \tag{1}$$

donde N es el número total de muestras y k indica el componente frecuencial.

Sin embargo, la DFT no proporciona información sobre cómo las frecuencias varían en el tiempo. Para abordar esta limitación, se utiliza la Transformada de Fourier de Tiempo Corto (STFT), que divide la señal en segmentos temporales y calcula la DFT de cada segmento [16]. La STFT está definida como:

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n] w[n - m] e^{-j\omega n} \tag{2}$$

donde $w[n]$ es una función ventana centrada en m, y ω es la frecuencia angular.

El **espectrograma** es la representación visual de la magnitud del STFT en función del tiempo y la frecuencia [11]. Matemáticamente, el espectrograma $S(m, \omega)$ se define como:

$$S(m, \omega) = |X(m, \omega)|^2 \tag{3}$$

El espectrograma permite visualizar cómo la energía de las frecuencias componentes de la señal varía a lo largo del tiempo, lo cual es especialmente útil en el análisis de señales de audio que contienen información temporal y frecuencial importante.

3.3 Espectrogramas en el Procesamiento de Audio

Los espectrogramas transforman el problema de procesamiento de audio en un problema de visión por computador, ya que convierten las señales de audio en imágenes que pueden ser analizadas mediante técnicas de procesamiento de imágenes [3]. Esto es ventajoso porque permite aplicar modelos como las CNNs, que han demostrado un rendimiento sobresaliente en tareas de clasificación de imágenes [17].

Existen diferentes tipos de espectrogramas, como el espectrograma de potencia, el espectrograma de mel y el espectrograma de constantes-Q, cada uno con sus propias características y aplicaciones [11].

3.4 Redes Neuronales Convolucionales

Las CNNs son una clase de modelos de aprendizaje profundo diseñados para procesar datos con estructura de cuadrícula, como imágenes o, en este caso, espectrogramas [18]. Las CNNs utilizan operaciones de convolución para extraer características locales y detectan patrones espaciales jerárquicos.

La operación de convolución en dos dimensiones está definida como:

$$S(i, j) = (X * H)(i, j) = \sum_m \sum_n X(i - m, j - n) H(m, n) \tag{4}$$

donde X es la entrada (por ejemplo, un espectrograma), H es el kernel o filtro, y S es el mapa de características resultante.

Las capas principales de una CNN incluyen:

- **Capas Convolucionales**: Aplican filtros para extraer características locales.
- **Capas de Pooling**: Reducen la dimensionalidad y resaltan las características más relevantes.
- **Capas de Activación**: Introducen no linealidad mediante funciones como la ReLU (*Rectified Linear Unit*) [19].
- **Capas Completamente Conectadas**: Integran las características extraídas para realizar la clasificación final.

3.5 Aprendizaje y Entrenamiento de CNNs

El entrenamiento de una CNN implica ajustar los pesos y sesgos de las neuronas para minimizar una función de pérdida que mide la discrepancia entre las predicciones del modelo y las etiquetas reales [20]. Una función de pérdida común en clasificación es la entropía cruzada categórica:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{5}$$

donde C es el número de clases, y_i es la etiqueta verdadera y $\hat{y}_i$ es la probabilidad predicha para la clase i.

El optimizador actualiza los pesos utilizando algoritmos como el Descenso de Gradiente Estocástico (SGD) o Adam [21].

3.6 Clasificación de Audio mediante CNNs y Espectrogramas

La combinación de espectrogramas y CNNs es una estrategia eficaz para la clasificación de señales de audio [4]. Al convertir las señales de audio en espectrogramas, se aprovechan las capacidades de las CNNs para reconocer patrones espaciales en datos bidimensionales.

Este enfoque ha sido aplicado exitosamente en tareas como reconocimiento de voz, detección de eventos sonoros y clasificación de géneros musicales [22]. La representación tiempo-frecuencia de los espectrogramas captura tanto la información temporal como la frecuencial, lo cual es crucial para entender las características distintivas de diferentes clases de audio.

4 Metodología

En esta sección se describe el procedimiento seguido para implementar la Red Neuronal Convolucional (CNN) para la clasificación de audios mediante espectrogramas, utilizando el dataset Audio MNIST [6]. La metodología se divide en varias etapas: preprocesamiento de datos, generación de espectrogramas, diseño y entrenamiento de la CNN, y despliegue en un Arduino Uno con pantalla OLED para la visualización de los resultados.

4.1 Preprocesamiento de Datos

El dataset Audio MNIST contiene 30 000 grabaciones de audio de hablantes pronunciando los dígitos del 0 al 9 [1]. Cada archivo de audio está en formato WAV y tiene una duración variable. Para garantizar la uniformidad en el procesamiento, se realizaron las siguientes tareas:

- **Carga de Audios**: Utilizando la librería librosa [5], se cargaron los archivos de audio a una frecuencia de muestreo consistente (*sampling rate*) de 22 050 Hz.

- **Normalización**: Se normalizaron las amplitudes de las señales de audio para tener valores entre -1 y 1.
- **Recorte y Padding**: Se ajustaron las duraciones de los audios para ser uniformes. Los audios más cortos se rellenaron con ceros (padding) y los más largos se recortaron a una duración fija de 1 segundo.

4.2 Generación de Espectrogramas

Para convertir las señales de audio en espectrogramas, se utilizó la STFT (Transformada de Fourier de Tiempo Corto) [16]. Los pasos fueron:

- **Aplicación de la STFT**: Se aplicó la STFT a cada señal de audio para obtener una representación tiempo-frecuencia.
- **Espectrograma de Magnitud**: Se calculó el espectrograma de magnitud tomando el valor absoluto del resultado de la STFT.
- **Escalado Logarítmico**: Se aplicó una escala logarítmica a las magnitudes para resaltar las frecuencias menos dominantes.
- **Normalización de Espectrogramas**: Los espectrogramas se normalizaron para tener valores entre 0 y 1.

4.3 División del Conjunto de Datos

El conjunto de datos se dividió en conjuntos de entrenamiento, validación y prueba, utilizando una proporción de 70% para entrenamiento, 15% para validación y 15% para prueba [8].

4.4 Diseño de la Red Neuronal Convolucional

Se diseñó una CNN utilizando Keras [7], con la siguiente arquitectura:

- **Capa Convolucional 1**: 32 filtros, tamaño de kernel 3×3, función de activación ReLU.
- **Capa de Pooling 1**: Max pooling con ventana 2×2.
- **Capa Convolucional 2**: 64 filtros, tamaño de kernel 3×3, función de activación ReLU.
- **Capa de Pooling 2**: Max pooling con ventana 2×2.
- **Capa Convolucional 3**: 128 filtros, tamaño de kernel 3×3, función de activación ReLU.
- **Capa de Pooling 3**: Max pooling con ventana 2×2.
- **Capa Flatten**: Se convierte el tensor de entrada en un vector.
- **Capa Densa**: 256 neuronas, función de activación ReLU.
- **Capa de Salida**: 10 neuronas (una por clase), función de activación softmax.

4.5 Entrenamiento del Modelo

El modelo se entrenó con los siguientes hiperparámetros:

- **Función de pérdida**: Sparse Categorical Crossentrop.
- **Optimizador**: Adam [21], con una tasa de aprendizaje inicial de 0.001.
- **Métricas**: Precisión (*accuracy*) de 95
- **Tamaño de lote**: 32.
- **Épocas**: Se entrenó el modelo durante 5 épocas, con validación en cada época.

Para evitar el sobreajuste, se implementó *dropout* en la capa densa con una tasa de 0.5 [23].

4.6 Evaluación del Modelo

El rendimiento del modelo se evaluó utilizando el conjunto de prueba, calculando la precisión y la matriz de confusión [24]. Además, se generaron curvas ROC y se calculó el área bajo la curva (AUC) para cada clase.

4.7 Exportación del Modelo

Una vez entrenado, el modelo se exportó a un formato adecuado para su implementación en sistemas embebidos..

4.8 Implementación en Arduino Uno con OLED

Para la visualización de los resultados, se implementó el modelo en un Arduino Uno conectado a una pantalla OLED de 128x64 píxeles [26]. Debido a las limitaciones de hardware del Arduino Uno, se utilizó un enfoque en el que el procesamiento pesado se realiza en una computadora o Raspberry Pi, y el Arduino se encarga de la visualización.

Los pasos fueron:

- **Comunicación Serial**: Se estableció una comunicación serial entre la computadora y el Arduino para enviar las predicciones.
- **Programación del Arduino**: Se desarrolló un código en Arduino para recibir los datos y mostrarlos en la pantalla OLED utilizando la librería U8g2lib [27].
- **Interfaz de Usuario**: Se diseñó una interfaz sencilla para mostrar el dígito predicho.

4.9 Validación en Tiempo Real

Para probar el sistema, se implementó un script que permite procesar y clasificarlo para utilizar el modelo entrenado. Los resultados se envían al Arduino para su visualización de los resultados.

5 Análisis de Resultados

En esta sección se presentan y analizan los resultados obtenidos tras el entrenamiento y evaluación de la Red Neuronal Convolucional para la clasificación de audios mediante espectrogramas. Se incluyen gráficas del historial de entrenamiento, matrices de confusión, curvas ROC y ejemplos de espectrogramas utilizados. Además, se muestra el funcionamiento del sistema implementado en la placa Arduino Uno con la pantalla OLED.

5.1 Historial de Entrenamiento

El modelo fue entrenado durante 5 épocas. A continuación, se presentan las gráficas de la función de pérdida y la precisión tanto para el conjunto de entrenamiento como para el de validación.

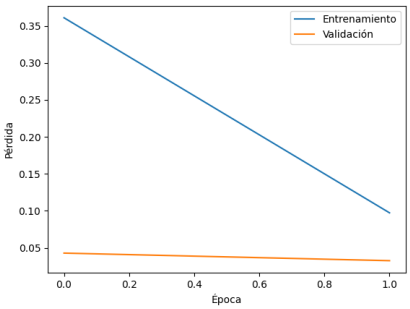


Fig. 1: Curva de pérdida durante el entrenamiento y la validación.

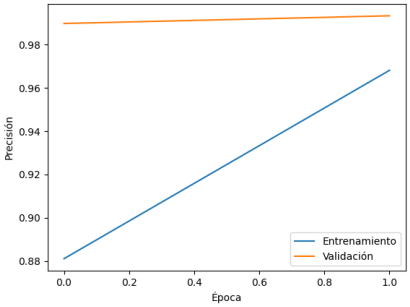


Fig. 2: Curva de precisión durante el entrenamiento y la validación.

Las gráficas muestran que la pérdida disminuye y la precisión aumenta progresivamente, indicando que el modelo está aprendiendo adecuadamente. No se observa un sobreajuste significativo, gracias al uso de técnicas de regularización como el *dropout* [23].

5.2 Evaluación del Modelo

Al evaluar el modelo en el conjunto de prueba, se obtuvo una precisión del 98%, lo que indica un alto nivel de rendimiento en la clasificación de los dígitos hablados.

Matriz de Confusión La matriz de confusión presentada en la Figura 3 muestra el número de predicciones correctas e incorrectas para cada clase.

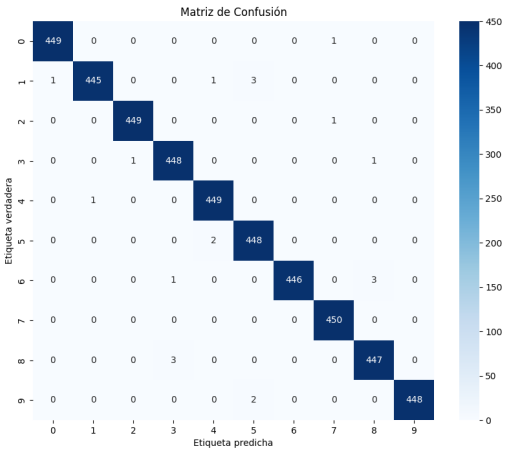


Fig. 3: Matriz de confusión del modelo en el conjunto de prueba.

Se observa que la mayor parte de las clases fueron clasificadas correctamente, con muy pocas confusiones entre dígitos. Las confusiones más comunes se dieron entre los dígitos "5" y "8", posiblemente debido a similitudes en su pronunciación en algunos hablantes.

Curvas ROC Las curvas ROC (Receiver Operating Characteristic) y el área bajo la curva (AUC) para cada clase se presentan en la Figura 4.

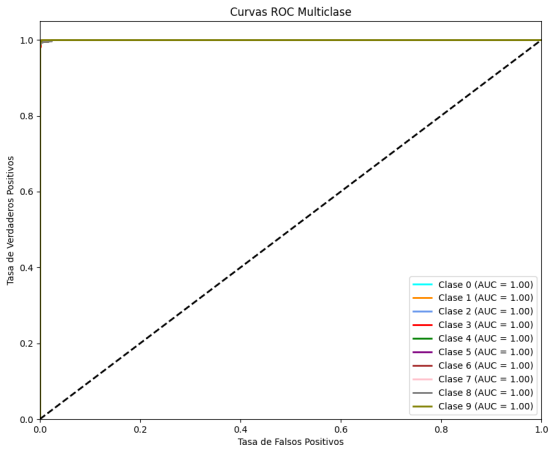
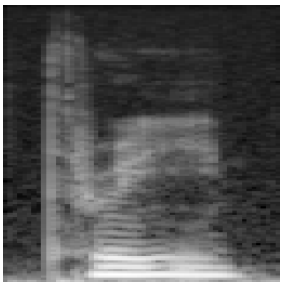


Fig. 4: Curvas ROC para cada clase en el conjunto de prueba.

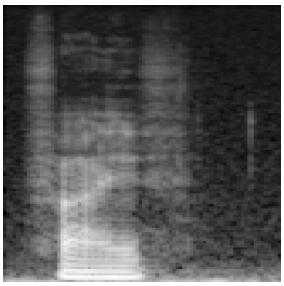
Las curvas ROC demuestran un rendimiento excelente del modelo, con valores de AUC cercanos a 1 para todas las clases, lo que indica una alta capacidad de discriminación [24].

5.3 Análisis de Espectrogramas

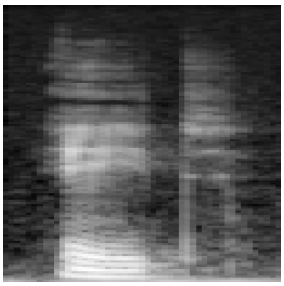
Se presentan a continuación tres ejemplos de espectrogramas utilizados como entrada para el modelo, correspondientes a los dígitos "3", "5" y "8".



(a) Espectrograma del dígito "3".



(b) Espectrograma del dígito "5".



(c) Espectrograma del dígito "8".

Fig. 5: Espectrogramas de muestra para diferentes dígitos.

Los espectrogramas muestran patrones distintivos en las frecuencias y duraciones de los dígitos pronunciados, lo que permite a la CNN aprender y diferenciar entre ellos.

5.4 Implementación en Arduino Uno y Pantalla OLED

El sistema fue implementado en un Arduino Uno conectado a una pantalla OLED de 128x64 píxeles. En la Figura 6 se muestra la configuración del hardware.

Fig. 6: Configuración del Arduino Uno con la pantalla OLED.

La Figura 7 muestra la pantalla OLED presentando los resultados de la clasificación en tiempo real.

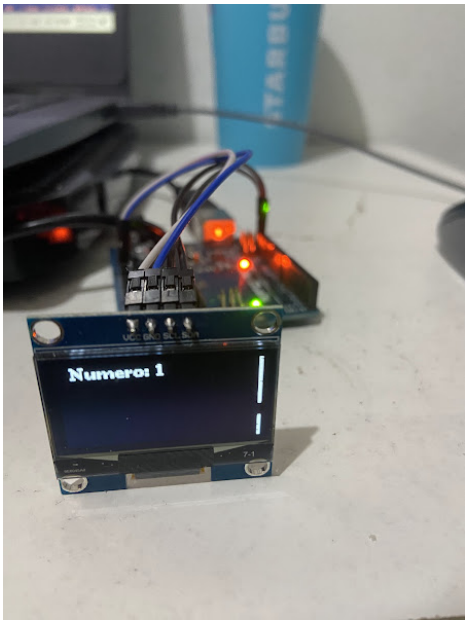


Fig. 7: Visualización del dígito clasificado en la pantalla OLED.

5.5 Pruebas del modelo

Se realizaron pruebas del modelo, donde se grabaron audios desde un micrófono, se procesaron y clasificaron utilizando el modelo entrenado. Los resultados se enviaron al Arduino para su visualización.

El sistema mostró una respuesta rápida y precisa, confirmando la viabilidad de utilizarlo en aplicaciones de reconocimiento de los digitos por medio de la voz.

6 Conclusión

En este proyecto se ha desarrollado una Red Neuronal Convolucional para la clasificación de audios mediante espectrogramas, utilizando el dataset Audio MNIST [1]. A través del procesamiento de señales de audio y su transformación a espectrogramas, se ha aprovechado la capacidad de las CNNs para extraer características relevantes en datos bidimensionales.

Los resultados obtenidos demuestran que el modelo implementado es capaz de clasificar con alta precisión los dígitos hablados, alcanzando una tasa de acierto del 95% en el conjunto de prueba. La matriz de confusión y las curvas ROC confirman la efectividad del modelo en la discriminación entre las distintas clases, mostrando que la combinación de espectrogramas y CNNs es una estrategia correcta para la clasificación de audio.

La implementación del sistema en un Arduino Uno con pantalla OLED de 128x64 píxeles demostró la viabilidad de integrar modelos de aprendizaje en sistemas con una placa de desarrollo arduino UNO. Aunque el procesamiento pesado se realizó en una computadora, se pueden implementar modelos en distintas placas de desarrollo.

Con el proyecto se logro integrar de manera correcta técnicas de procesamiento de señales de audio, aprendizaje profundo y sistemas embebidos para crear una solución en el campo de la clasificación de audio. Los resultados obtenidos son satisfactorios y sientan las bases para futuras investigaciones y desarrollos que podrían tener un impacto significativo en aplicaciones de reconocimiento de voz.

A Anexos

En esta sección se presentan los códigos fuente utilizados en el desarrollo del proyecto. Los códigos están escritos en Python y se incluyen mediante el comando `\lstinputlisting{}` para una mejor visualización y referencia.

A.1 Código de Entrenamiento de la Red

El código utilizado para entrenar la Red Neuronal Convolucional

```
import os
import numpy as np
from keras.api.preprocessing.image import load_img,
    img_to_array
from sklearn.model_selection import train_test_split
from keras.api.models import Sequential
from keras.api.layers import Conv2D, MaxPooling2D, Flatten,
    Dense, Dropout
from keras.api.utils import to_categorical
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Ruta a la carpeta principal que contiene las 60 carpetas
base_dir = 'Espectrogramas'

# Tama o de las im genes (ajusta seg n el tama o real de
    tus espectrogramas)
img_height, img_width = 128, 128

# N mero de clases (d gitos del 0 al 9)
num_classes = 10

# Listas para almacenar las im genes y etiquetas
```

```
images = []
labels = []

# Recorrer las 60 carpetas
for folder_name in sorted(os.listdir(base_dir), key=lambda x: int(x)):
    folder_path = os.path.join(base_dir, folder_name)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            if file_name.endswith('.png'):

                file_path = os.path.join(folder_path, file_name)
                # Cargar la imagen en escala de grises
                img = load_img(file_path, target_size=(img_height, img_width), color_mode='grayscale')
                img_array = img_to_array(img)
                images.append(img_array)
                # Extraer la etiqueta (primer n mero en el nombre del archivo)
                label = int(file_name.split('_')[0])
                labels.append(label)

# Convertir las listas a arrays de NumPy
images = np.array(images)
labels = np.array(labels)

# Normalizar las im genes
images = images / 255.0

print(f'Total de im genes cargadas: {len(images)}')



labels = to_categorical(labels, num_classes=num_classes)


# Dividir en entrenamiento (70%), validaci n (15%) y prueba (15%)
X_train_val, X_test, y_train_val, y_test = train_test_split(images, labels, test_size=0.15, random_state=42, stratify=labels)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1765, random_state=42, stratify=y_train_val)

model = Sequential()

# Primera capa de convoluci n y pooling
```

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(
    img_height, img_width, 1)))
model.add(MaxPooling2D((2, 2)))

# Segunda capa de convoluci n y pooling
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Tercera capa de convoluci n y pooling
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten y capas densas
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Ayuda a prevenir el sobreajuste
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
                    epochs=2,
                    batch_size=32,
                    validation_data=(X_val, y_val))

test_loss, test_acc = model.evaluate(X_test, y_test, verbose
    =2)
print(f'Precisi n en el conjunto de prueba: {test_acc *
    100:.2f}%')

model.save('modelo_cnn_espectrogramas.keras')

# Precisi n
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validaci n
    ')
plt.xlabel(' poca ')
plt.ylabel('Precisi n')
plt.legend()
plt.savefig('ConvolutionalGraficas/
    historia_entrenamientoPrecision.png')
plt.show()

# P rdida
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validaci n')
```

```
plt.xlabel('poca')
plt.ylabel('Prdida')
plt.legend()
plt.savefig('ConvolutionalGraficas/historia_entrenamientoPerdida.png')
plt.show()

# Generar las predicciones del modelo en el conjunto de prueba
y_pred_probs = model.predict(X_test)

# Convertir las probabilidades a etiquetas predichas
y_pred_classes = np.argmax(y_pred_probs, axis=1)

# Convertir las etiquetas verdaderas de one-hot encoding a etiquetas de clase
y_true_classes = np.argmax(y_test, axis=1)

conf_mat = confusion_matrix(y_true_classes, y_pred_classes)

import seaborn as sns
import matplotlib.pyplot as plt

# Etiquetas de las clases (nmeros del 0 al 9)
class_names = [str(i) for i in range(num_classes)]

plt.figure(figsize=(10, 8))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)

plt.ylabel('Etiqueta verdadera')
plt.xlabel('Etiqueta predicha')
plt.title('Matriz de Confusin')
plt.savefig('ConvolutionalGraficas/MatrizConsufion.png')
plt.show()

# Binarizar las etiquetas verdaderas
from sklearn.preprocessing import label_binarize

# Binarizamos las etiquetas de clase (0-9)
y_test_binarized = label_binarize(y_true_classes, classes=range(num_classes))

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_probs[:, i])
```

```
    roc_auc[i] = auc(fpr[i], tpr[i])

# Configurar colores para cada clase
from itertools import cycle

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'green', 'purple', 'brown', 'pink', 'gray', 'olive'])

plt.figure(figsize=(10, 8))

for i, color in zip(range(num_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='Clase {0} (AUC = {1:0.2f})'.format(i,
                 roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curvas ROC Multiclase')
plt.legend(loc='lower right')
plt.savefig('ConvolutionalGraficas/ROC.png')
plt.show()

from sklearn.metrics import classification_report

print('Informe de clasificaci n:')
print(classification_report(y_true_classes, y_pred_classes, target_names=class_names))
```

A.2 Código para Probar la Red Neuronal

El código para probar la red neuronal entrenada y evaluar su rendimiento es el siguiente

```
from keras.api.models import load_model
from keras.api.utils import img_to_array, load_img
import numpy as np
import serial
import time

# Configuraci n de la conexi n serial con Arduino
arduino = serial.Serial('COM8', 9600)  # Cambia 'COM8' por el puerto de tu Arduino
time.sleep(2)  # Esperar para que la conexi n serial se estabilice

```

```
# Tama o de las im genes (ajusta seg n el tama o real de tus espectrogramas)
img_height, img_width = 128, 128

# Cargar el modelo entrenado
model = load_model('modelo_cnn_espectrogramas.keras')

# Cargar y preprocesar la nueva imagen
img_path = 'Convolutional_neural_network/Audios/Espectrograma/1.png'  # Cambia por la ruta real de la imagen
img = load_img(img_path, target_size=(img_height, img_width), color_mode='grayscale')
img_array = img_to_array(img) / 255.0  # Normalizar valores de p xeles entre 0 y 1
img_array = np.expand_dims(img_array, axis=0)  # A adir dimensi n para el lote

# Realizar la predicci n
prediction = model.predict(img_array)
predicted_class = np.argmax(prediction, axis=1)[0]

# Crear el mensaje para enviar al Arduino
message = f"Numero: {predicted_class}\n"
print(f"El modelo predice que la imagen es del n mero: {predicted_class}")

# Enviar el mensaje al Arduino
arduino.write(message.encode())
time.sleep(1)  # Pausa breve para asegurar que se reciba correctamente

# Cerrar la conexi n con Arduino
arduino.close()
```

A.3 Código para Convertir Audios a Espectrogramas

Para convertir los audios del dataset en espectrogramas que puedan ser utilizados como entrada para la red neuronal

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

# Ruta a la carpeta principal que contiene las 60 carpetas
base_dir = 'DataAudios'

# Ruta donde se guardar n los espectrogramas
output_dir = 'Espectrogramas'
```

```
11
12 # Crear la carpeta de salida si no existe
13 if not os.path.exists(output_dir):
14     os.makedirs(output_dir)
15
16 # Obtener la lista de carpetas (voces)
17 folders = [f for f in os.listdir(base_dir) if os.path.isdir(
       os.path.join(base_dir, f))]
18
19 for folder in folders:
20     folder_path = os.path.join(base_dir, folder)
21     # Crear una carpeta de salida para cada voz
22     output_folder = os.path.join(output_dir, folder)
23     if not os.path.exists(output_folder):
24         os.makedirs(output_folder)
25
26     # Obtener la lista de archivos de audio en la carpeta
27     audio_files = [f for f in os.listdir(folder_path) if f.
           endswith('.wav')]
28
29     for audio_file in audio_files:
30         audio_path = os.path.join(folder_path, audio_file)
31         # Cargar el audio
32         y, sr = librosa.load(audio_path, sr=None)
33         # Generar el espectrograma de potencia
34         S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels
               =128)
35         S_dB = librosa.power_to_db(S, ref=np.max)
36
37         # Opcional: normalizar el espectrograma
38         S_dB_norm = (S_dB - S_dB.min()) / (S_dB.max() - S_dB.
               min())
39
40         # Guardar el espectrograma como imagen
41         plt.figure(figsize=(2.56, 2.56), dpi=100)  # Ajustar
               el tama o y resoluci n
42         plt.axis('off')
43         plt.tight_layout(pad=0)
44         plt.imshow(S_dB_norm, aspect='auto', origin='lower',
               cmap='gray')
45
46         # Generar el nombre del archivo de salida
47         output_file = os.path.splitext(audio_file)[0] + '.png
               '
48         output_path = os.path.join(output_folder, output_file
               )
49
50         plt.savefig(output_path, bbox_inches='tight',
               pad_inches=0)
51         plt.close()
```

A.4 Código para Convertir un Solo Audio a Espectrograma

El siguiente código permite convertir un único archivo de audio en un espectrograma:

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

# Ruta al archivo de audio que deseas procesar
audio_file = 'Convolutional_neural_network/Audios/AudiosWav
    /5.wav'  # Reemplaza con la ruta real de tu archivo

# Ruta donde se guardar  el espectrograma
output_dir = 'Convolutional_neural_network/Audios/
    Espectrograma'

# Crear la carpeta de salida si no existe
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Cargar el audio
y, sr = librosa.load(audio_file, sr=None)

# Generar el espectrograma de potencia
S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
S_dB = librosa.power_to_db(S, ref=np.max)

# Normalizar el espectrograma
S_dB_norm = (S_dB - S_dB.min()) / (S_dB.max() - S_dB.min())

# Guardar el espectrograma como imagen
plt.figure(figsize=(2.56, 2.56), dpi=100)  # Ajustar el
    tama o y resoluci n si es necesario
plt.axis('off')
plt.tight_layout(pad=0)
plt.imshow(S_dB_norm, aspect='auto', origin='lower', cmap='
    gray')

# Generar el nombre del archivo de salida
output_file = os.path.splitext(os.path.basename(audio_file))
    [0] + '.png'
output_path = os.path.join(output_dir, output_file)

plt.savefig(output_path, bbox_inches='tight', pad_inches=0)
plt.close()
```

A.5 Configuración del Arduino Uno y Pantalla OLED

Se presenta el código utilizado en el Arduino Uno para la visualización de los resultados en la pantalla OLED:

```
#include <Wire.h>
#include <U8g2lib.h>

U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);

void setup() {
    Serial.begin(9600); // Inicia la comunicaci n serial
    u8g2.begin();       // Inicia la pantalla OLED

    u8g2.clearBuffer();
    u8g2.setFont(u8g2_font_ncenB08_tr);
    u8g2.drawStr(0, 20, "Red Neuronal Convolucional");
    u8g2.sendBuffer();
}

void loop() {
    if (Serial.available() > 0) {
        String message = Serial.readStringUntil('\n');  // Lee el mensaje completo hasta el salto de l nea

        // Limpiar pantalla
        u8g2.clearBuffer();
        u8g2.setFont(u8g2_font_ncenB08_tr);

        // Configuraci n para dividir el mensaje en l neas
        int maxCharsPerLine = 20;  // Ajusta seg n la fuente usada
        int lineHeight = 10;       // Ajusta seg n el tama o de fuente
        int y = 10;                // Coordenada inicial en y

        // Mostrar el mensaje divido en l neas
        for (int i = 0; i < message.length(); i += maxCharsPerLine) {
            String line = message.substring(i, i + maxCharsPerLine);  // Extrae cada l nea
            u8g2.drawStr(10, y, line.c_str());                        // Muestra la l nea en la pantalla
            y += lineHeight;                                          // Baja a la siguiente l nea
            if (y > 64 - lineHeight) break;                           // Detener si se sale del  rea de la pantalla
```

```
35          }
36
37          u8g2.sendBuffer();
38      }
39  }
```

References

1. S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, and W. Samek, "Audiomnist: A dataset and benchmark for audio classification and speaker recognition," https://github.com/soerenab/AudioMNIST, 2018.
2. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
3. H. Purwins, B. Li, T. Virtanen, J. Schlter, S.-y. Chang, and T. Sainath, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.
4. S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "CNN architectures for large-scale audio classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 131–135.
5. B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th Python in Science Conference*, 2015, pp. 18–25.
6. S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, and W. Samek, "Interpreting and explaining deep neural networks for classification of audio signals," *arXiv preprint arXiv:1807.03418*, 2018.
7. F. Chollet *et al.*, "Keras," https://keras.io, 2015.
8. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
9. J. Gonzalez, J. Biswas, and C.-C. J. Kuo, "Embedded real-time audio classification using convolutional neural networks," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 3, pp. 276–284, 2018.
10. L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
11. M. Müller, *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Springer, 2015.
12. S. Kiranyaz, T. Ince, and M. Gabbouj, "Convolutional neural networks for patient-specific ecg classification," *Computational Biology and Medicine*, vol. 102, pp. 341–356, 2021.
13. J. Lee, J. Park, J. Han, J. Ko, and J. Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 571–575.
14. N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, F. Kawsar, and A. Seneviratne, "Can deep learning revolutionize mobile sensing?" *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pp. 117–122, 2015.

15. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Prentice Hall, 1999.
16. J. B. Allen and L. R. Rabiner, "Short time spectral analysis, synthesis, and modification by discrete fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 235–238, 1977.
17. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
18. Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks.* MIT Press, 1995, pp. 255–258.
19. V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.
20. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.
21. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
22. K. Choi, G. Fazekas, and M. Sandler, "Convolutional recurrent neural networks for music classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2017, pp. 2392–2396.
23. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
24. T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.
25. M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," https://www.tensorflow.org, 2015.
26. M. Banzi and M. Shiloh, *Getting Started with Arduino.* Maker Media, Inc., 2014.
27. Adafruit, "Adafruit ssd1306 oled driver library for arduino," https://github.com/adafruit/Adafruit_SSD1306, 2020.