

IA's en el diario vivir

Kevin Jesgreg Cañón Alvarez
kevin.canon@mail.escuelaing.edu.co
PROM_M, maestria en ciencia de datos
Escuela Colombiana de Ingeniería Julio Garavito

Título de tu informe

En el siguiente se presenta una pequeña demostración de las distintas herramientas con las que contamos actualmente para desarrollar cualquier tarea del cotidiano gracias al desarrollo de la tecnología y las ias, por ello se presentan 4 diferentes aplicaciones en las cuales se puede observar la gran ventaja que estas ofrecen para el avance humano.

AUDIO_SAMPLES

Mediante el uso de ChatGPT es ahora posible programar de manera más eficiente, para este caso se quiso diseñar un divisor de audios para una tarea específica, aumentar la cantidad de muestras de un audio para crear un mejor dataset que reconociera ese tipo de audio:



You

buenas tardes chatgpt, quisiera un código en python al cuál yo le pueda adjuntar un archivo de audio y como resultado obtenga 10 fragmentos diferentes de audio que provengan del original, con una duracion cada uno de 15 segundos

Aquí tienes un ejemplo de código para lograrlo:

python

Copy code

```
from pydub import AudioSegment
import os

def split_audio(input_file, output_dir, num_fragments, fragment_duration):
    audio = AudioSegment.from_file(input_file)
    total_duration = len(audio)

    if total_duration < (num_fragments * fragment_duration):
        print("El archivo de audio es demasiado corto para dividirlo en tantos fragmentos")
        return

    for i in range(num_fragments):
        start_time = i * (total_duration / num_fragments)
        end_time = (i + 1) * (total_duration / num_fragments)
        fragment = audio[start_time:end_time]
        fragment.export(os.path.join(output_dir, f"fragment_{i + 1}.wav"), format="wav")

if __name__ == "__main__":
    input_file = "tu_archivo_de_audio.mp3" # Reemplaza con la ruta de tu archivo de audio
    output_dir = "fragments" # Carpeta de salida para los fragmentos
    num_fragments = 10
    fragment_duration = 15 * 1000 # Duración en milisegundos

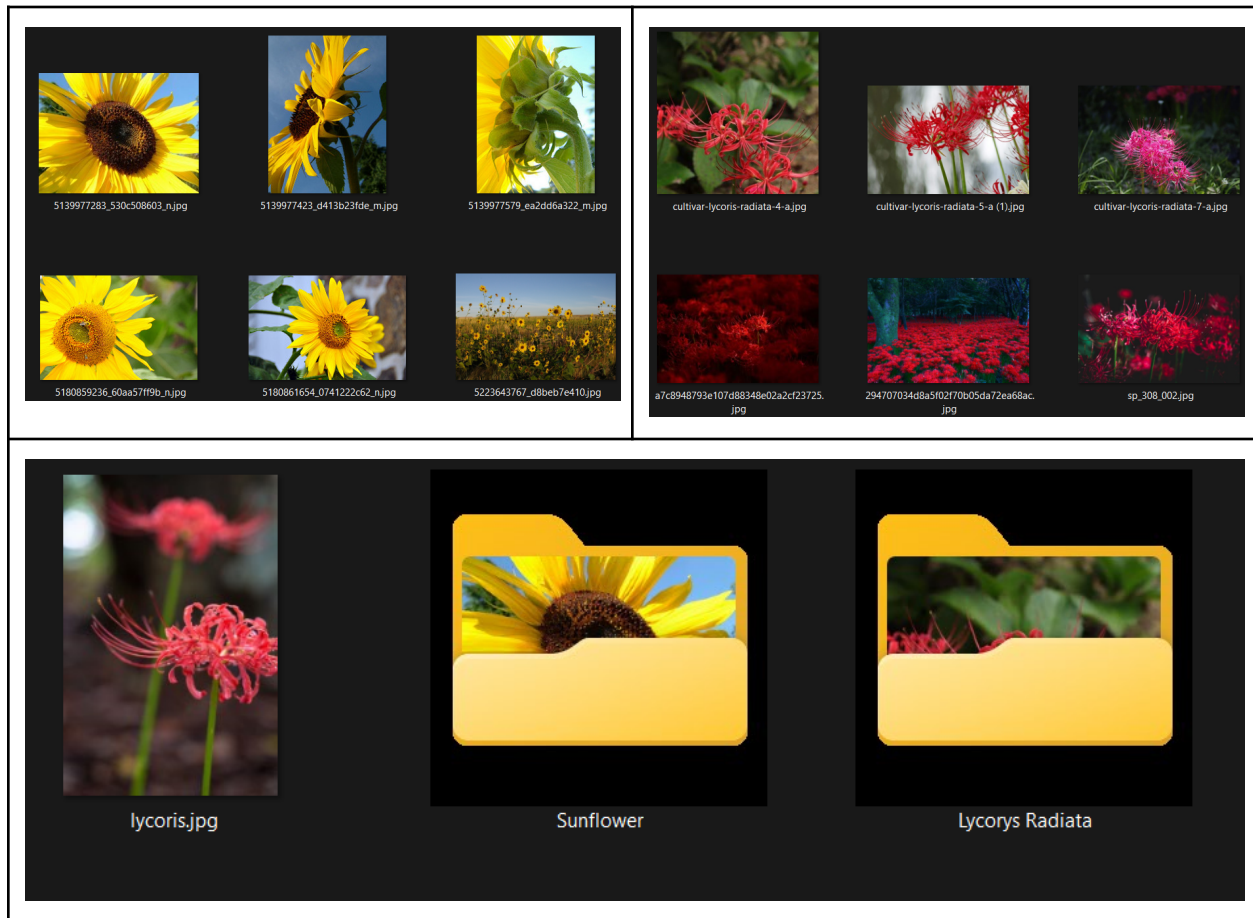
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)

    split_audio(input_file, output_dir, num_fragments, fragment_duration)
```

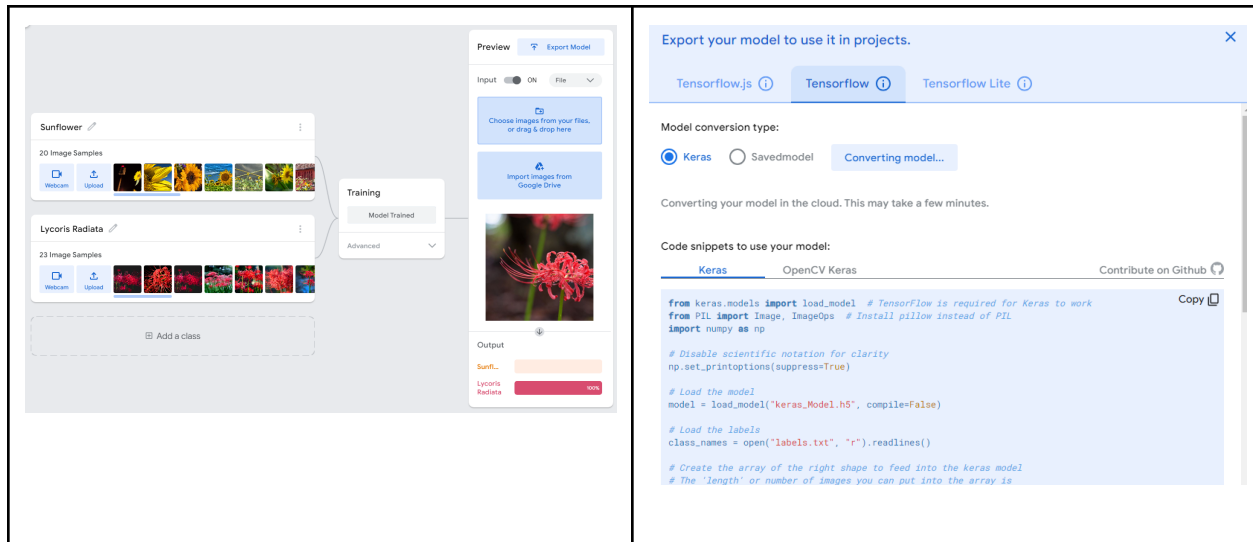
FLOWER_RECOGNITION

Una aplicación que permite aumentar nuestro conocimiento momentáneo acerca de un tema, por ejemplo, acerca de flores, gracias al reconocimiento de imágenes es posible entrenar un modelo que diferencia entre distintos tipos de flor

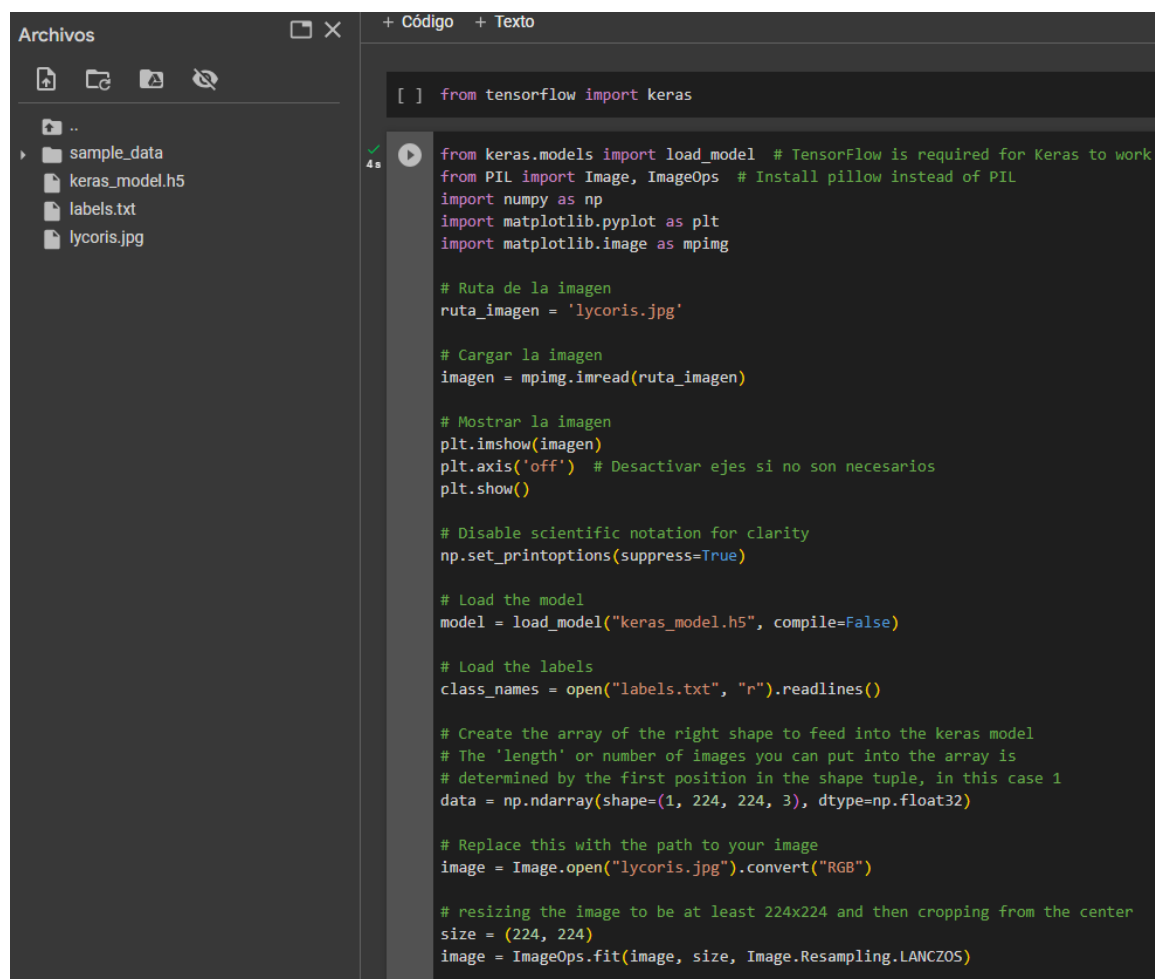
para realizarlo, se requiere crear un data set con fotografías acerca de cada categoría deseada, para esta IA se creó un repertorio de fotos de girasoles y de la flor lycoris radiata



Mediante el software de Teachable Machine ofrecido por Google, es posible crear un modelo de reconocimiento de imágenes, que diferencie entre estas dos categorías de flores:



Se comprobó el funcionamiento del modelo, y se exportó a Tensorflow Keras, se obtuvo el código en python, el archivo del modelo (keras_model.h5) y las etiquetas de las categorías (labels.txt) para poder trabajarlo en colab:



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory named 'sample_data' containing files 'keras_model.h5', 'labels.txt', and 'lycoris.jpg'. The code editor contains Python code that imports TensorFlow and Keras, loads the model and labels, and processes the image 'lycoris.jpg'.

```
[ ] from tensorflow import keras

from keras.models import load_model # TensorFlow is required for Keras to work
from PIL import Image, ImageOps # Install pillow instead of PIL
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Ruta de la imagen
ruta_imagen = 'lycoris.jpg'

# Cargar la imagen
imagen = mpimg.imread(ruta_imagen)

# Mostrar la imagen
plt.imshow(imagen)
plt.axis('off') # Desactivar ejes si no son necesarios
plt.show()

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

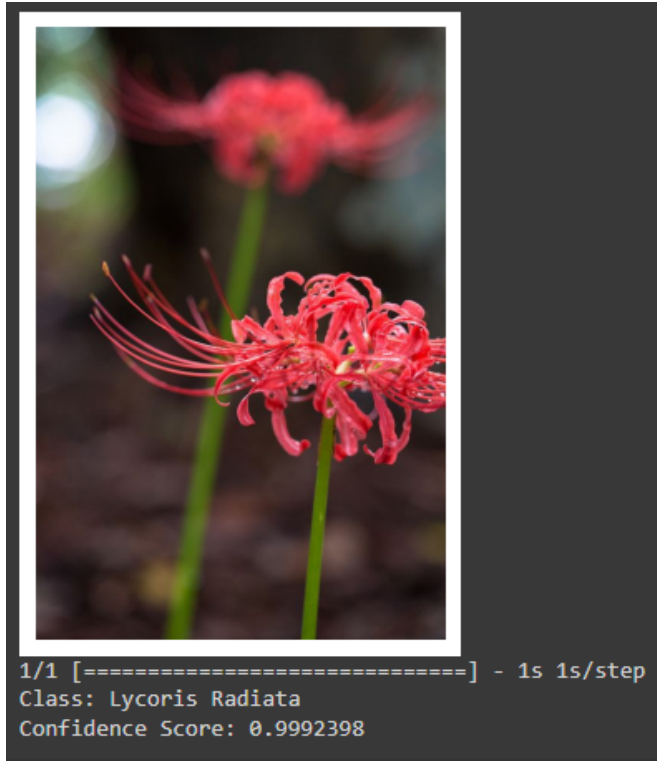
# Load the model
model = load_model("keras_model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Replace this with the path to your image
image = Image.open("lycoris.jpg").convert("RGB")

# resizing the image to be at least 224x224 and then cropping from the center
size = (224, 224)
image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)
```



Se obtiene el valor de la clase hallada y su probabilidad

MUSICGEN

Este es un modelo recreado de hugging face, el cuál permite crear muestras de música dadas unas características de entrada como lo son género, emociones, frecuencia, instrumentos, entre otros

```
You
explicame línea por línea el siguiente código y dime como y que alternativas tengo para
personalizar el código:
!python3 -m pip install -U git+https://github.com/facebookresearch/audiocraft#egg=audiocraft
# !python3 -m pip install -U audiocraft
from audiocraft.models import musicgen
from audiocraft.utils.notebook import display_audio
import torch
model = musicgen.MusicGen.get_pretrained('medium', device='cuda')
model.set_generation_params(duration=8)
res = model.generate([
    'crazy EDM, heavy bang',
    'classic reggae track with an electronic guitar solo',
    'lofi slow bpm electro chill with organic samples',
    'rock with saturated guitars, a heavy bass line and crazy drum break and fills',
    'earthy tones, environmentally conscious, ukulele-infused, harmonic, breezy, easygoing, organic
instrumentation, gentle grooves',
],
    progress=True)
display_audio(res, 32000)
```

VOICE_COMMANDS

Una aplicación bastante útil es el procesamiento de audios para reconocer la voz y crear comandos de palabras, para este caso se desea construir un modelo que interpretara instrucciones de control para un reproductor de musica con las palabras play, stop, next, back y shuffle, para ello se le pidió a ChatGPT mediante un ejemplo ya dado que ajustara el código para las especificaciones deseadas:

```

You
hola, el siguiente codigo entrena una inteligencia artificial para reconocer si un audio es un
ronquido o no, deseo que en vez de eso el codigo quede adaptado para que reconozca entre 6
diferentes tipos de palabras, las cuales son: play, stop, next, back, shuffle y empty, a cada palabra
se le asignará una carpeta de entrenamiento y de prueba para que el modelo quede entrenado
adecuadamente, cómo quedaría el codigo completo corregido como lo deseo?
from tensorflow import keras
from tensorflow.keras import layers
import os
import librosa
import numpy as np
from librosa.util import normalize

# Definir listas para almacenar espectrogramas y etiquetas
spectrogram_list = []
labels_list = []

# Carpeta de ronquidos (entrenamiento)
ronquidos_folder = '/content/drive/MyDrive/CAYD/ronqui'
# Carpeta de no ronquidos (entrenamiento)
no_ronquidos_folder = '/content/drive/MyDrive/CAYD/noronqui'

```

**ChatGPT**

Para adaptar el código para reconocer seis palabras diferentes ("play", "stop", "next", "back", "shuffle", y "empty"), puedes seguir estos pasos:

1. Organiza tus datos de entrenamiento y prueba en carpetas separadas para cada palabra. Asigna una carpeta para cada palabra, y dentro de cada carpeta, coloca los archivos de audio correspondientes.
2. Modifica el código para cargar los datos de entrenamiento y prueba de acuerdo con las nuevas carpetas y etiquetas.
3. Ajusta el modelo para tener 6 neuronas en la capa de salida y use la función de activación softmax, ya que ahora estamos tratando con un problema de clasificación multiclase.

Aquí tienes el código modificado:

```
python Copy code

from tensorflow import keras
from tensorflow.keras import layers
import os
import librosa
import numpy as np
from librosa.util import normalize

# Definir listas para almacenar espectrogramas y etiquetas
spectrogram_list = []
labels_list = []

# Lista de palabras
palabras = ["play", "stop", "next", "back", "shuffle", "empty"]

# Carpeta de entrenamiento
train_folder = '/content/drive/MyDrive/CAYD/train'

# Recorre las carpetas de entrenamiento
for i, palabra in enumerate(palabras):
    folder_path = os.path.join(train_folder, palabra)

    # Recorre los archivos de audio en cada carpeta
    for filename in os.listdir(folder_path):
        if filename.endswith('.wav'):
            filepath = os.path.join(folder_path, filename)
            audio, _ = librosa.load(filepath, sr=None)
            spectrogram = librosa.feature.melspectrogram(y=audio, sr=44100)
            spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
            # Ajusta la longitud a 128
            if spectrogram.shape[1] < 128:
                pad_width = 128 - spectrogram.shape[1]
                spectrogram = np.pad(spectrogram, pad_width=((0, 0), (0, pad_width)),
                    else:
                        spectrogram = spectrogram[:, :128]
            spectrogram_list.append(spectrogram)
            labels_list.append(1) # Etiqueta 1 para la palabra actual

# Convierte las listas de espectrogramas y etiquetas en matrices de datos
X_train = np.array(spectrogram_list)
y_train = np.array(labels_list)

# Añadir una dimensión adicional para el canal
X_train = X_train[:, :, :, np.newaxis]
```

```
# Define el modelo de aprendizaje automático
model = keras.Sequential([
    layers.Input(shape=(128, 128, 1)),
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(8, activation='softmax') # Se cambió a 8 neuronas y activación softmax
])

# Compila el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])

# Entrena el modelo
model.fit(X_train, y_train, epochs=15)

# Guarda el modelo
model.save('/content/drive/MyDrive/CAYD/tu_modelo.h5')
```

Este código asume que tienes la siguiente estructura de carpetas:

```
markdown Copy code

- train
- play
  - archivo1.wav
  - archivo2.wav
  ...
- stop
  - archivo1.wav
  - archivo2.wav
  ...
- next
  ...
- back
  ...
- shuffle
  ...
- empty
  ...
```

como respuesta chatgpt brinda un código implementando todo lo deseado, etiquetando parte por parte lo que este realiza y explicando procesos adicionales, en este caso la organización del data set para entrenar el modelo, así que optó por subir los archivos a un drive con el fin de poder acceder a ellos mejor desde colab


Mi unidad > PROM ▼


Tipo ▼


Personas ▼

Modificado ▼

Nombre

 train

 test

 voice.h5


Mi unidad > PROM > train ▼


Tipo ▼

Personas ▼


Modificado ▼


Nombre


 shuffle

 next

 play

 empty

 back

 pause


Mi unidad > PROM > test ▼


Tipo ▼


Personas ▼


Modificado ▼


Nombre


 back

 empty

 shuffle

 pause

 next

 play

Con el código dado al modelo le faltaba recorrer los archivos de prueba



You

ok, puedo visualizar que recorre la carpeta de entrenamiento, pero y la definicion para leer la carpeta de prueba?



ChatGPT

Mis disculpas por el descuido. Aquí está el código para cargar los datos de prueba de manera similar a como se hizo con los datos de entrenamiento:

```
python Copy code

# Carpeta de prueba
test_folder = '/content/drive/MyDrive/CAYD/test'

# Definir listas para almacenar espectrogramas y etiquetas de prueba
spectrogram_test_list = []
labels_test_list = []

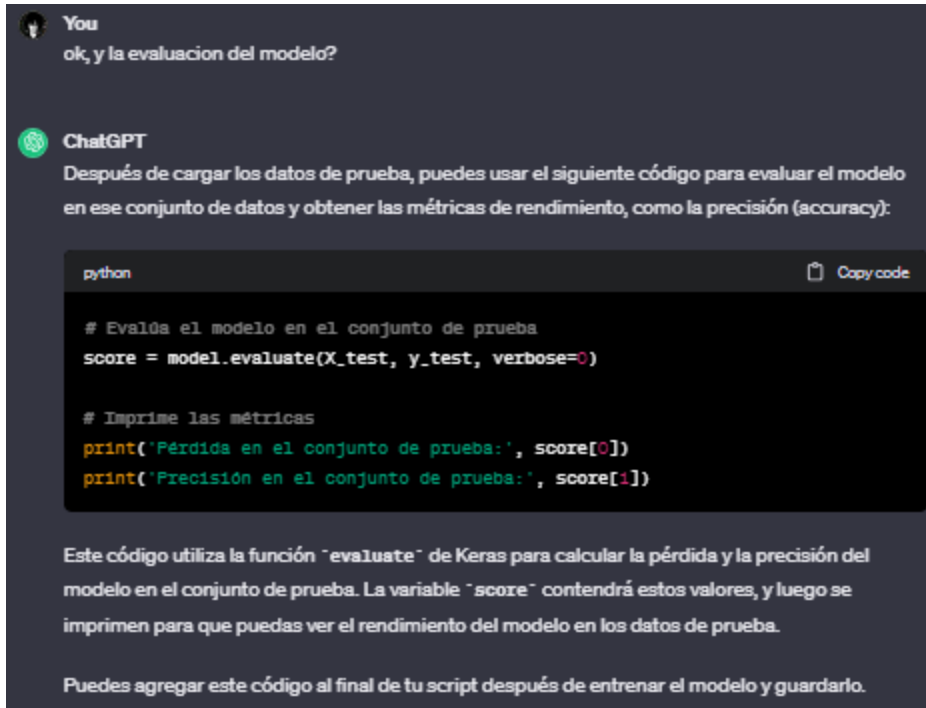
# Recorre las carpetas de prueba
for i, palabra in enumerate(palabras):
    folder_path = os.path.join(test_folder, palabra)

    # Recorre los archivos de audio en cada carpeta
    for filename in os.listdir(folder_path):
        if filename.endswith('.wav'):
            filepath = os.path.join(folder_path, filename)
            audio, _ = librosa.load(filepath, sr=None)
            spectrogram = librosa.feature.melspectrogram(y=audio, sr=44100)
            spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
            # Ajusta la longitud a 128
            if spectrogram.shape[1] < 128:
                pad_width = 128 - spectrogram.shape[1]
                spectrogram = np.pad(spectrogram, pad_width=((0, 0), (0, pad_width))),
            else:
                spectrogram = spectrogram[:, :128]
            spectrogram_test_list.append(spectrogram)
            labels_test_list.append(i) # Etiqueta 1 para la palabra actual

# Convierte las listas de prueba en matrices de datos
X_test = np.array(spectrogram_test_list)
y_test = np.array(labels_test_list)

# Añadir una dimensión adicional para el canal
X_test = X_test[:, :, :, np.newaxis]
```

Luego le fue solicitado la evaluación del modelo para conocer la pérdida y la precisión del modelo



al compilar el código se empezaban a presentar errores, pero preguntando al chat fue posible comprender rápidamente el error y obtener la versión del código que funcionaría sin problemas:



You

explicame el siguiente error, como lo puedo solucionar? como seria el codigo corregido o que debo cambiar?

```
13     try:
14         do_return = True
--> 15         retval = ag__converted_call(ag__ld(step_function), (ag__ld(self),
ag__ld(iterator)), None, fscope)
16     except:
17         do_return = False
```

ValueError: in user code:



ChatGPT

El error que estás experimentando indica que hay una discrepancia en las formas ("shape") de las predicciones ("logits") y las etiquetas ("labels") durante el cálculo de la pérdida. En tu modelo, parece que estás utilizando una función de pérdida binaria ("binary_crossentropy"), pero ahora tu problema es de clasificación multiclase, por lo que deberías usar una función de pérdida para clasificación multiclase.

En tu caso, ya has modificado la capa de salida de tu modelo para tener 6 neuronas y utilizas la activación softmax, lo cual es correcto para un problema de clasificación multiclase. Ahora necesitas ajustar la función de pérdida.

Cambia la siguiente línea en la compilación de tu modelo:

python

Copy code

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

a:

python

Copy code

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc
```

La función de pérdida "sparse_categorical_crossentropy" es adecuada para problemas de clasificación multiclase cuando las etiquetas son enteros. Esto debería resolver el problema que estás enfrentando. Asegúrate de hacer esta modificación antes de entrenar tu modelo.

You
explicame que significa la siguiente linea de codigo y los parametros que se encuentran dentro de ella:
`model.fit(X_train, y_train, epochs=15)`

ChatGPT
La línea de código `model.fit(X_train, y_train, epochs=15)` es responsable de entrenar el modelo de red neuronal. Aquí hay una explicación de cada uno de los parámetros:

1. **"X_train"**: Este es el conjunto de datos de entrada de entrenamiento. Contiene los espectrogramas de audio que se utilizarán para entrenar el modelo.
2. **"y_train"**: Estas son las etiquetas correspondientes a los datos de entrada de entrenamiento. Cada etiqueta indica la clase a la que pertenece el espectrograma de audio. En tu caso, parece que estas etiquetas son números enteros que representan las diferentes palabras que deseas reconocer.
3. **"epochs=15"**: Este parámetro indica la cantidad de veces que el modelo pasará por todo el conjunto de entrenamiento. En este caso, estás configurando el número de épocas en 15, lo que significa que el modelo se entrenará durante 15 iteraciones completas a través de tus datos de entrenamiento.

Durante el proceso de entrenamiento, la red neuronal ajusta sus pesos y sesgos para minimizar la función de pérdida entre las predicciones del modelo y las etiquetas reales. Cada época implica un pase completo a través de todos los datos de entrenamiento.

En resumen, `model.fit` es el método que inicia el proceso de entrenamiento de tu modelo de red neuronal. Utiliza los datos de entrada (`X_train`) junto con las etiquetas correspondientes (`y_train`) para ajustar los parámetros internos del modelo a fin de realizar predicciones más precisas. El número de épocas (`epochs`) determina cuántas veces se repetirá este proceso de ajuste.

You
necesito ahora probar mi modelo con un audio cualquiera, asi que debo cargar un archivo de audio y necesito que me indique en cual de las 6 categorias se encuentra el audio, si es play, pause, next, back, shuffle y empty

luego se le pregunta cómo se puede probar el modelo creado con un archivo de audio cualquiera



ChatGPT

Para probar tu modelo con un archivo de audio, puedes seguir estos pasos:

1. Cargar y procesar el archivo de audio de la misma manera que lo hiciste durante la preparación de los datos de entrenamiento y prueba. Esto incluye la carga del archivo, la extracción de características (como el espectrograma), y la adaptación de la longitud si es necesario.
2. Utilizar el modelo entrenado para hacer predicciones sobre el nuevo archivo de audio.
3. Interpretar la salida del modelo y asignar la categoría correspondiente a la clase predicha.

Aquí hay un ejemplo básico de cómo podrías hacerlo:

python

Copy code

```
import librosa
import numpy as np

# Cargar el archivo de audio que deseas probar
audio_path = 'ruta_del_archivo/audio_prueba.wav'
audio, _ = librosa.load(audio_path, sr=None)

# Procesar el archivo de audio (similar a la preparación de datos de entrenamiento)
spectrogram = librosa.feature.melspectrogram(y=audio, sr=44100)
spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
if spectrogram.shape[1] < 128:
    pad_width = 128 - spectrogram.shape[1]
    spectrogram = np.pad(spectrogram, pad_width=((0, 0), (0, pad_width)), mode='const')
else:
    spectrogram = spectrogram[:, :128]

# Añadir una dimensión adicional para el canal
input_data = spectrogram[np.newaxis, :, :, np.newaxis]

# Hacer la predicción con el modelo
predictions = model.predict(input_data)

# Obtener la clase predicha
predicted_class = np.argmax(predictions)

# Mapear la clase predicha a la categoría correspondiente
categorias = ["play", "pause", "next", "back", "shuffle", "empty"]
categoria_predicha = categorias[predicted_class]

# Imprimir la categoría predicha
print(f'El audio se clasifica como: {categoria_predicha}')
```

adecuar el código para que procese el audio e indique a qué palabra se refiere



You

tengo el siguiente código para cargar un audio y reconocer si es un ronquido o no, como sería el código adecuándolo para las nuevas categorías posibles?



ChatGPT

Entiendo que ya tienes un código para cargar un audio y reconocer si es un ronquido o no, y ahora quieres adaptarlo para reconocer las nuevas categorías ("play", "pause", "next", "back", "shuffle", y "empty"). Aquí hay una adaptación básica que puedes realizar:

```
python
import librosa
import numpy as np
from tensorflow import keras

# Cargar el modelo entrenado
model = keras.models.load_model('/content/drive/MyDrive/CAYD/tu_modelo.h5')

# Cargar el archivo de audio que deseas probar
audio_path = 'ruta_del_archivo/audio_prueba.wav'
audio, _ = librosa.load(audio_path, sr=None)

# Procesar el archivo de audio (similar a la preparación de datos de entrenamiento)
spectrogram = librosa.feature.melspectrogram(y=audio, sr=44100)
spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
if spectrogram.shape[1] < 128:
    pad_width = 128 - spectrogram.shape[1]
    spectrogram = np.pad(spectrogram, pad_width=((0, 0), (0, pad_width)), mode='constant')
else:
    spectrogram = spectrogram[:, :128]

# Añadir una dimensión adicional para el canal
input_data = spectrogram[np.newaxis, :, :, np.newaxis]

# Hacer la predicción con el modelo
predictions = model.predict(input_data)

# Obtener la clase predicha
predicted_class = np.argmax(predictions)

# Mapear la clase predicha a la categoría correspondiente
categorias = ["play", "pause", "next", "back", "shuffle", "empty"]
categoria_predicha = categorias[predicted_class]

# Imprimir la categoría predicha
print(f'El audio se clasifica como: {categoria_predicha}')
```

como el data set creado fue almacenado en drive, se le solicitó a chatgpt cómo podría integrar los archivos del drive al proyecto en colab y reproducir el audio ingresado para comprobar que realmente es la palabra que se pronuncia

You
como sería la línea de código que me permita importar un archivo de audio de mi drive en colab y que lo reproduzca?

ChatGPT
En Google Colab, puedes cargar archivos directamente desde Google Drive. Aquí hay un ejemplo de cómo puedes hacerlo y reproducir un archivo de audio:

```
python
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from IPython.display import Audio

# Autenticación para acceder a Google Drive
gauth = GoogleAuth()
gauth.LocalWebserverAuth()
drive = GoogleDrive(gauth)

# ID del archivo de audio en Google Drive
file_id = 'ID_DEL_ARCHIVO' # Reemplaza 'ID_DEL_ARCHIVO' con el ID real de tu archivo

# Descargar el archivo de audio desde Google Drive
downloaded_file = drive.CreateFile({'id': file_id})
downloaded_file.GetContentFile('audio_prueba.wav') # Guardar el archivo con el nombre

# Reproducir el archivo de audio
Audio('audio_prueba.wav')
```

Con este resultado final se logro obtener el código corregido con las características deseadas para correr en colab, obteniendo como resultado el código voice_commands, el modelo voice.h5

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 126, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 63, 63, 32)	0
flatten_1 (Flatten)	(None, 127008)	0
dense_1 (Dense)	(None, 6)	762054
Total params: 762374 (2.91 MB)		
Trainable params: 762374 (2.91 MB)		
Non-trainable params: 0 (0.00 Byte)		

Arquitectura del modelo

```

# Añadir una dimensión adicional para el canal
input_data = spectrogram[np.newaxis, :, :, np.newaxis]

# Hacer la predicción con el modelo
predictions = model.predict(input_data)

# Obtener la clase predicha
predicted_class = np.argmax(predictions)

# Mapear la clase predicha a la categoría correspondiente
categorias = ["play", "pause", "next", "back", "shuffle", "empty"]
categoria_predicha = categorias[predicted_class]

# Imprimir la categoría predicha
print(f'El audio se clasifica como: {categoria_predicha}')

```

```

1/1 [=====] - 0s 48ms/step
El audio se clasifica como: play

```

Predicción de un audio

CONCLUSIÓN

Mediante los anteriores proyectos se puede visualizar el gran alcance que le permite a la tecnología mejorar nuestra eficiencia en trabajos y realizar cualquier tarea, desde programación hasta entrenar modelos para identificar cosas, siendo chatgpt una de las herramientas más poderosas para nuestro diario, que brinda la posibilidad de adaptar la gran variedad de modelos que han sido creados a nuestro gusto.

