Instituto Tecnológico y de Estudios Superiores de Monterrey

Proyecto Integrador

Profesor: Dra. Grettel Barceló Alonso

Instrucciones de voz

Equipo - 38

Rubén Marcos Ramos Guerrero - A01793131

Edgar Lopez Valdes - A01339939

Eddie Guadalupe Elorza Ruiz- A01793547

19 de Mayo del 2024

Pruebas con Tensorflow

Para probar con un modelo creado en tensorflow, el primer paso fue separar los audios en casos positivos y negativos.

```
WAKE_WORD_DIRECTORY = "Audios/HolaTecBot"

BACKGROUND_NOISE_DIRECTORY = "Audios/HolaCasoNegativo"
```

Se creó un modelo de DNN en la función create_model para el reconocimiento de wake words, debido a su capacidad de aprender y reconocer patrones complejos en datos de audio. Este modelo en particular utiliza una arquitectura de red neuronal convolucional (CNN) con capas Conv1D y MaxPooling1D para extraer características relevantes del audio de manera eficiente. La combinación de estas capas permite al modelo captar variaciones sutiles en las señales de audio, mejorando la precisión en la detección de la palabra clave.

```
def create_model(input_shape):

   print("Creating Model")

   model = tf.keras.Sequential([

       layers.Input(shape=input_shape),

       layers.Conv1D(64, 3, activation='relu'),

       layers.MaxPooling1D(2),

       layers.Conv1D(128, 3, activation='relu'),

       layers.MaxPooling1D(2),

       layers.Flatten(),

       layers.Dense(64, activation='relu'),

       layers.Dense(1, activation='sigmoid')

   ])

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

```
    return model
```

El modelo se entrenó con varios hiperparametros, entre 20 y 80 epochs tratando de encontrar un balance, desafortunadamente la reducida cantidad de datos disponibles. Solo juntamos alrededor de 50 audios nos llevan a pensar que el modelo pudiese estar sobreentrenado. Agregamos etiquetas a los dataframes de manera que fuera más explícito si el valor que debe ser asociado a un audio es positivo o negativo.

```
def train_model():

    print("Training")

    X_positive = load_audio_data(WAKE_WORD_DIRECTORY)

    X_negative = load_audio_data(BACKGROUND_NOISE_DIRECTORY)

    Y_positive = np.ones(len(X_positive))  # Etiquetas para datos positivos

    Y_negative = np.zeros(len(X_negative)) # Etiquetas para datos negativos



    X = np.concatenate([X_positive, X_negative])

    X = np.expand_dims(X, axis=-1)

    Y = np.concatenate([Y_positive, Y_negative])



    indices = np.arange(len(X))

    np.random.shuffle(indices)

    X = X[indices]

    Y = Y[indices]



    input_shape = X.shape[1:]

    model = create_model(input_shape)

    model.fit(X, Y, epochs=30, batch_size=32)
```

```
model.save("wake_word_detection_model.keras")
```

Al probar el modelo con un pequeño set que se guardó en el directorio de pruebas, los resultados no han sido positivos hasta el momento. Obtuvimos un 100% de identificación de wake word. En todos los audios nos devolvió un match, esto desafortunadamente no es lo que se esperaba. Consideramos que el volumen y calidad de datos son factores muy importantes para el desempeño pobre del modelo.

```
def test_model_with_audio_files(test_directory):
    model = tf.keras.models.load_model('wake_word_detection_model.keras')
    preprocessed_audio_data  = load_audio_data(test_directory)

    for i, audio_sample in enumerate(preprocessed_audio_data):
        audio_sample = audio_sample.reshape(1, -1)
        prediction = model.predict(audio_sample)

        if prediction > 0.5:
            print(f"Audio file {i+1}: Wake word detected!")
        else:
            print(f"Audio file {i+1}: No wake word detected.")
```

Librería Porcupine prueba exitosa

Porcupine es una herramienta especializada en la detección de “wake words”. Utiliza redes neuronales profundas y tiene la bondad de que una vez se han cargado los modelos de manera local, pueden ejecutarse offline y con bajo uso de recursos.

En la prueba realizada encontramos dos momentos clave, donde se precarga el modelo, como limitante, encontramos que solo ciertas palabras forman parte de su librería pero con palabras similares se logró obtener resultados muy positivos. Durante la configuración se tuvo que obtener un Api Key para llamar el servicio como se muestra en la sección de abajo.

```
try:
    # Inicialización de Porcupine con parámetros específicos
    porcupine = pvporcupine.create(
        access_key=os.environ['PORCUPINE_ACCESS_KEY'],  # Clave de acceso a Porc
        keyword_paths=['./Hola-te-bot_es_mac_v3_0_0.ppn'],  # Ruta del archivo d
        model_path='./porcupine_params_es.pv',  # Ruta del modelo de Porcupine
        sensitivities=[0.9]  # Sensibilidad del detector de palabras clave
    )
```

```
    while True:
        # Lectura del flujo de audio
        pcm = audio_stream.read(porcupine.frame_length)
        pcm = struct.unpack_from("h" * porcupine.frame_length, pcm)

        # Procesamiento del audio para detectar la palabra clave
        keyword_index = porcupine.process(pcm)

        if keyword_index >= 0:
            print(f"{datetime.now()} - 'Hola TecBot' detectado")
```

Una vez configurado, se abre el canal de escucha en el equipo local, en este caso el micrófono de la computadora. Se compara el input de audio del canal y se procesa para tratar de encontrar las “wake words”.

De manera general se obtuvieron resultados positivos por este camino, el programa reconoce de manera efectiva las *wake words* “Hola TecBot”

WIREFRAMES

El desarrollo de Tec Bot ha sido impulsado por la necesidad de ofrecer un soporte técnico de alta calidad y una experiencia de usuario optimizada. Con estos wireframes, queremos mostrarles la estructura básica y el flujo de interacción del chatbot, destacando las funcionalidades clave y la lógica detrás de cada una de sus respuestas.