Reconocimiento del lenguaje natural

Eduardo Farinango, Marlon Tuquerrez, Bryan Tandazo, Escuela Politécnica Nacional (EPN), Quito - Ecuador

***Resumen* – En este documento se presentan los lineamientos básicos para la implementación de algoritmos basados en el reconocimiento automático del habla utilizando SPICE de TensorFlow, siendo esto la base del algoritmo debido a que en su programación esta un modelo entrenado que nos permitirá interactuar mediante tonos de voz. También se implementara algoritmos que junto a SpeechRecongnition podremos transcribir lo que estamos hablando y con la ayuda de Google Colab pondremos en marcha los algoritmos y observar como nuestra voz se transforma a tonos musicales y texto.**

I. Introducción

En el presente informe se hablara acerca de cómo podemos interactuar con una computadora mediante el uso de inteligencia artificial, sabiendo que el lenguaje de una maquina no es el mismo de una persona buscaremos implementar un puente que permita esta comunicación, este puente son las redes neuronales que con sus algoritmos de análisis y entrenamiento nos permitirán en pocas palabras hablar con una computadora. Existen diferentes lenguajes de programación que son aptos para la implementación de inteligencia artificial este trabajo se enfocara en su implementación en Python para poder traducir nuestra a voz a tonos musicales y poder también transcribir en tiempo real lo que estamos diciendo.

II. Conocimiento teórico

Para nuestro propósito en esta aplicación de la Inteligencia artificial para poder interactuar con una computadora mediante algoritmos pre entrenados necesitaremos varias herramientas y conocimiento medio en programación en Python que es un lenguaje de alto nivel y de código abierto que busca tener un código simple y limpio y que permite implementar algoritmos de distintos tipos de inteligencia artificial [1]. Como segunda herramienta utilizaremos Google Colab que nos permitirá escribir código de Python en el navegador gracias a su especialización en aprendizaje automático y por qué nos brinda la facilidad de utilizar sus recursos como base [1]. Para pode realizar el procesamiento del lenguaje natural utilizaremos más servicios de google en este caso utilizaremos Tensorflow que es una biblioteca de código abierto para poder realizar aprendizaje automático [1]. Utilizaremos Spice que es un modelo musical pre entrenado [1] y Specch Recognition que nos ayudara a reconocer el habla de un humano y permite que una computadora nos entienda [2].

III. Desarrollo del contenido

A. Librerías

Para poder cumplir con nuestro propósito junto a las herramientas que nos permiten implementar código necesitaremos la instalación de lagunas librerías para el procesamiento del lenguaje natural dentro de Google Colab instalaremos las siguientes librerías: numba en su versión 0.48 y librosa-music esto nos permitirá utilizar de manera eficaz nuestro modelo entrenado Spice como se puede observar en la *Fig **1***.

```
pip install pydub numba==0.48 librosa music21

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Collecting numba==0.48
  Downloading numba-0.48.0-1-cp37-cp37m-manylinux2014_x86_64.whl (3.5 MB)
     |████████████████████████████████| 3.5 MB 6.4 MB/s
Requirement already satisfied: librosa in /usr/local/lib/python3.7/dist-packages (0.8.1)
Requirement already satisfied: music21 in /usr/local/lib/python3.7/dist-packages (5.5.0)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from numba==0.4
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from numba==0.48
Collecting llvmlite<0.32.0,>=0.31.0dev0
  Downloading llvmlite-0.31.0-cp37-cp37m-manylinux1_x86_64.whl (20.2 MB)
     |████████████████████████████████| 20.2 MB 1.5 MB/s
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in /usr/local/lib/python3.7/dist-package
```

Fig 1. Instalación de librerías en Python

Una vez instalada las librerías de forma externar utilizaremos las facilidades de Colab, importaremos librerías principales que ya vienen preinstaladas.

TABLA 1
LIBERIAS PARA RECONOCIMIENTO DEL LENGUAJE NATURAL

Librería	**Uso**
Tensorflow	Biblioteca
Tensorflow_Hub	Repositorio de modelos entrenados
Numpy	Biblioteca funciones matemática.
Matplotlib	Generador de gráficos
Librosa	Análisis de audio
Math	Función matemática
Statistics	Funciones estadísticas
Ipython	Shell
Scipy	Algoritmos matemáticos
Base 64	Codificación
Music 21	Análisis musical Javascript
Pydub	Conversión de formatos

Instaladas e importadas las librerías tenemos lo necesario para poder iniciar la codificación del proceso de grabación y reconocimiento de voz.

B. *Ingreso de información.*

Procederemos con la grabación de nuestra voz para este punto debemos tomar en cuenta que lo estamos haciendo en una página web, para esto debemos utilizar un poco de javascript que nos permita activar nuestro micrófono para poder grabar nuestra melodía. Un punto importante es saber que el producto final de este proceso son melodías entonces debemos grabarnos cantando.

En la *Fig* ***2*** podemos observar que guardaremos nuestra grabación en una variable "RECORD" la cual tendrá un tiempo de 30 segundos de duración, esta se almacenará hasta que sea llamada en una función.

```
RECORD = """
const sleep  = time => new Promise(resolve => setTimeout(resolve, time))
const b2text = blob => new Promise(resolve => {
  const reader = new FileReader()
  reader.onloadend = e => resolve(e.srcElement.result)
  reader.readAsDataURL(blob)
})
var record = time => new Promise(async resolve => {
  stream = await navigator.mediaDevices.getUserMedia({ audio: true })
  recorder = new MediaRecorder(stream)
  chunks = []
  recorder.ondataavailable = e => chunks.push(e.data)
  recorder.start()
  await sleep(time)
  recorder.onstop = async ()=>{
    blob = new Blob(chunks)
    text = await b2text(blob)
    resolve(text)
  }
  recorder.stop()
})
"""

def record(sec=30):
  try:
    from google.colab import output
  except ImportError:
    print('No possible to import output from google.colab')
    return ''
  else:
    print('Recording')
    display(Javascript(RECORD))
    s = output.eval_js('record(%d)' % (sec*1000))
    fname = 'recorded_audio.wav'
    print('Saving to', fname)
    b = b64decode(s.split(',')[1])
    with open(fname, 'wb') as f:
      f.write(b)
    return fname
```

Fig 2. Grabación de audio.

Una vez grabada nuestra voz debemos procesarla con la ayuda de librerías de google ingresamos nuestra muestra al proceso de reconocimiento del lenguaje natural. En la *Fig* ***3*** se observa los mensajes de error en el caso de que la grabación no se realice correctamente.

```
#@title Archivo de entrada de audio

INPUT_SOURCE = 'RECORD'

print('You selected', INPUT_SOURCE)

if INPUT_SOURCE == 'RECORD':
  uploaded_file_name = record(13)
else:
  print('Unrecognized input format!')
  print('Please select "RECORD", "UPLOAD", or specify a file hosted on Google Drive or a file from the web to dow

You selected RECORD
Recording
Saving to recorded_audio.wav
```

Fig 3. Almacenamiento de audio para proceso

Nuestra voz grabada ingresa con ciertas características como frecuencias de onda y canales para nuestra práctica necesitaremos cambiar a una onda que funcione a 16khz y que solo tenga un canal mono que es una forma primitiva de reproducir sonido y va en una sola salida. En la fig 4 podemos ver la función que mediante el uso de una librería permite realizar esta acción.

```
EXPECTED_SAMPLE_RATE = 16000

def convert_audio_for_model(user_file, output_file='converted_audio_file.wav'):
  audio = AudioSegment.from_file(user_file)
  audio = audio.set_frame_rate(EXPECTED_SAMPLE_RATE).set_channels(1)
  audio.export(output_file, format="wav")
  return output_file
```

Fig 4. Cambio de frecuencia de onda.

Posterior a esto cargamos nuestro audio modificado en una variable sabiendo que este es el audio que se va a procesar, extraemos las características para verificar los pasos anteriores y mostramos en pantalla la forma de onda de nuestra voz como se observa en la fig 5.

```
sample_rate, audio_samples = wavfile.read(converted_audio_file, 'rb')
duration = len(audio_samples)/sample_rate
print(f'Sample rate: {sample_rate} Hz')
print(f'Total duration: {duration:.2f}s')
print(f'Size of the input: {len(audio_samples)}')
Audio(audio_samples, rate=sample_rate)

Sample rate: 16000 Hz
Total duration: 13.00s
Size of the input: 207934
▶ 0:00 / 0:12

[ ]  # Visualizando la onda del audio

     _ = plt.plot(audio_samples)
```

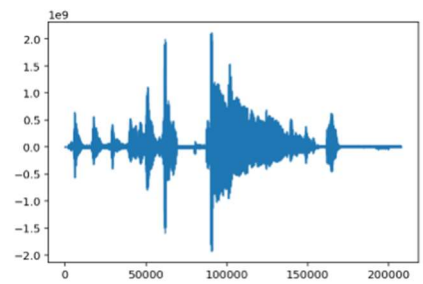


Fig 5. Carga del audio y grafica del audio.

Ahora procederemos a ejecutar nuestro modelo predictivo de Tensorflow el cual nos servirá para procesar nuestro audio tratado, la forma de trabajar de Spice es en base a tonos e

incertidumbres, para cargar nuestro modelo pre entrenado basta una línea de código:

model = hub.load("https://tfhub.dev/google/spice/2")

Para poder realizar una predicción obligatoriamente necesitamos datos de entrada los cuales ya tenemos que es nuestra voz grabada, el modelo pre entrenado el cual recibirá los datos y los procesara, para ver qué tan acertada es nuestra predicción en base los datos ingresados procederemos a trabajar con la librería matplotlib que nos permitirá graficar nuestros datos en un rango entre 0 y 1.

Como se observa en la *Fig* ***6*** nuestra grabación en forma de grafico que representa los tonos y la incertidumbre refiriéndose a la complejidad para poder realizar la predicción dentro del rango seleccionado.

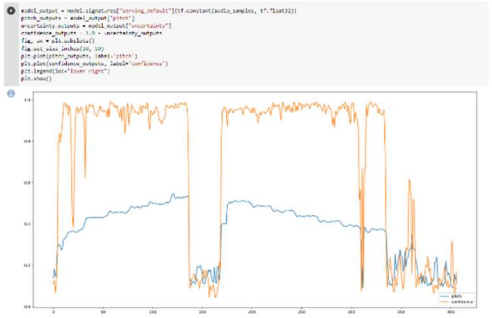


Fig 6. Niveles de incertidumbre y tonos

Para poder observar de mejor manera se procederá a realizar dos gráficos más, uno que nos muestra de mejor manera el espectro de nuestro audio en este punto lo deseamos en blanco y negro y lo sobrepondremos con el gráfico de nuestros datos entrenados.

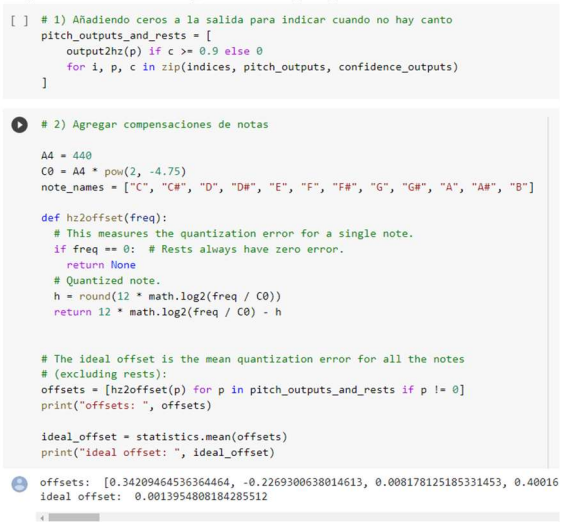
Fig 7. Predicción del modelo vs data ingresada

En la *Fig 7* vemos que la predicción se pinta de color rojo y está dentro de nuestra muestra inicial esto quiere decir que estamos dentro de un 90% de acercarnos a los datos de entrada y conseguiremos los tonos más cercanos a nuestra voz

C. Conversión a notas musicales

Para este paso debemos tomar en cuanta varias características tanto de las notas musicales y de la muestra resultante del entrenamiento.

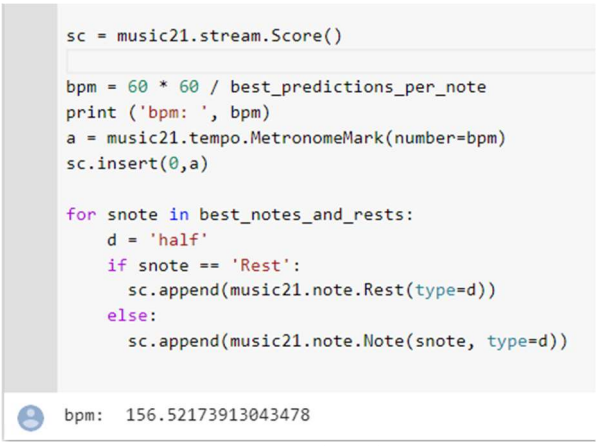
Una nota musical por lo general tiene 4 tiempos serian 4 segundos de duración, así que se procederá a agregar ceros cuando no exista sonido y compensar el tiempo de sonido de nuestra grabación con las notas musicales. Se ingresan las notas musicales en una variables y se procede a realizar los cálculos matemáticos para que pueda coincidir nuestros dos campos. Como se observa en la *Fig* ***8*** como resultado tenemos los desfaces y el valor ideal para este propósito.



```
[ ] # 1) Añadiendo ceros a la salida para indicar cuando no hay canto
    pitch_outputs_and_rests = [
        output2hz(p) if c >= 0.9 else 0
        for i, p, c in zip(indices, pitch_outputs, confidence_outputs)
    ]

    # 2) Agregar compensaciones de notas

    A4 = 440
    C0 = A4 * pow(2, -4.75)
    note_names = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]

    def hz2offset(freq):
      # This measures the quantization error for a single note.
      if freq == 0:  # Rests always have zero error.
        return None
      # Quantized note.
      h = round(12 * math.log2(freq / C0))
      return 12 * math.log2(freq / C0) - h


    # The ideal offset is the mean quantization error for all the notes
    # (excluding rests):
    offsets = [hz2offset(p) for p in pitch_outputs_and_rests if p != 0]
    print("offsets: ", offsets)

    ideal_offset = statistics.mean(offsets)
    print("ideal offset: ", ideal_offset)

    offsets:  [0.34209464536364464, -0.2269300638014613, 0.008178125185331453, 0.40016
    ideal offset:  0.0013954808184285512
```

Fig 8. Acercado el modelo a la grabación

Ahora se debera estimar la seuencia mas probable para que coincidad con los datos ingresado en ste caso la grabacion de nuestra voz, se cuantificara que valor se va a cuantificar para compensar el sesgo y s ebuscara el valor del error as bajo tomando como base un octavo lo minimo en el tiempo musical. Luego de la cuantificación se creara unscore en bpm y se ajustara la velocidad para lograr una sincornizacion adecuada con la grabación. Como se observa en la el algoritmo nos da como reusltado un bpm de 156.521.



```
sc = music21.stream.Score()

bpm = 60 * 60 / best_predictions_per_note
print ('bpm: ', bpm)
a = music21.tempo.MetronomeMark(number=bpm)
sc.insert(0,a)

for snote in best_notes_and_rests:
    d = 'half'
    if snote == 'Rest':
      sc.append(music21.note.Rest(type=d))
    else:
      sc.append(music21.note.Note(snote, type=d))

bpm:  156.52173913043478
```

Fig 9. Velocidad de predicción.

Ahora nuevamente con la ayuda de librerias de Javascript mostraremos las notas conseguidas mediante el score, la

visualización se la realiza mediante un pentagrama el cual tendra las notas musicales que coincidieron con nuestra voz.

De proceso anterior se consiguieron las siguientes coincidencias generales: F, A, C, D.

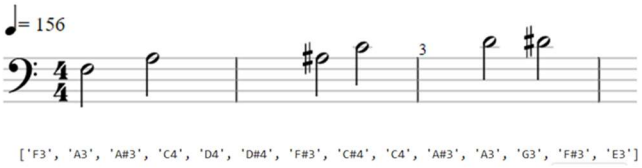


Fig 10. Pentagrama de notas en base a predicción.

D. Transformando notas a MIDI

Con el uso de las librerías importadas al inicio pasaremos nuestro entrenamiento junto a los scores conseguidos y transformados en notas musicales adecuadas a nuestra voz, las notas, musicales se convertirán en sonido y podremos comparar el original con el resultado.

```
converted_audio_file_as_midi = converted_audio_file[:-4] + '.mid'
fp = sc.write('midi', fp=converted_audio_file_as_midi)
```

```
wav_from_created_midi = converted_audio_file_as_midi.replace(' ', '_') + "_midioutput.wav"
print(wav_from_created_midi)
```

```
converted_audio_file.mid_midioutput.wav
```

Una vez tranformado en MIDI pasaremos al formato WAV el cual se podra escuchar en el navegador utilizando la librería timidit.

```
!timidity $converted_audio_file_as_midi -Ow -o $wav_from_created_midi
```

```
Playing converted_audio_file.mid
MIDI file: converted_audio_file.mid
Format: 1  Tracks: 1  Divisions: 1024
Sequence:
Playing time: ~14 seconds
Notes cut: 0
Notes lost totally: 0
```

```
Audio(wav_from_created_midi)
```

IV. CONCLUSIONES

Como se pudo observar en la aplicación de Detección por tono, cumple con los pasos del aprendizaje automático. Primero se ingresa los datos en este caso maneja sonidos, luego, se utiliza un modelo de entrenamiento ya entrenado en este caso sería SPICE, y finalmente con ese modelo se ingresa el sonido de prueba y como predicción nos muestra las notas musicales.

De la misma manera en el caso de Detección por voz, maneja los sonidos como datos de entrada, y utiliza un modelo entrenado que en este caso es SpeechRecognition y como predicción muestra el texto de lo que se dijo en la grabación.

Se pudo aplicar y entender que existen diferentes modelos los cuales ya están entrenados y que permiten realizar diferentes acciones, por tanto, nosotros solo tendríamos que proporcionarle datos entrada y el modelo exclusivamente se encarga de presentar las predicciones o los resultados.

El tema de Inteligencia Artificial como sabemos abarca muchos campos de la ciencia y uno de ellos es la forma en que nos comunicamos se puede concluir que si existe una forma de cambiar la voz y convertirla en tonos secuénciales y entendibles, existe también la forma de convertir la IA en aplicaciones que permitan facilitar a las personas su diario vivir, es decir puede ayudar a tomar decisiones de manera mas sencilla y sin perder mucho tiempo.

V. REFERENCIAS

[1] Tensorflow, «Tensorflow,» Tensorflow, [En línea]. Available: https://www.tensorflow.org/. [Último acceso: 31 08 2022].

[2] B. Lutkevich, «SearchCustomerExperience,» SearchCustomerExperience, 13 09 2021. [En línea]. Available: https://www.techtarget.com/searchcustomerexperience/definition/speech-recognition. [Último acceso: 31 08 2022].

[3] P. S. Foundation, «Python,» Python Software Foundation , [En línea]. Available: https://www.python.org/doc/. [Último acceso: 31 08 2022].

VI. BIOGRAFÍAS

Bryan Tandazo, nació en QuitoEcuador el 15 de mayo de 2000. Realizó sus estudios secundarios en la Unidad Educativa Bicentenario D7. Actualmente estudia en la Escuela Politécnica Nacional en la carrera Tecnología Superior en Desarrollo de Software y está cursando el cuarto semestre. Áreas de interés: Back End Developer, Inteligencia artificial, y soporte técnico.
(bryan.tandazo@epn.edu.ec)

Eduardo Farinango, nació en Quito -Ecuador en el año 1987. Realizo sus estudios secundarios en el sector público, pero se graduó en el Centro Educativo Asambleas de Dios, luego de dejar sus estudios universitarios en el año 2015 vuelve a reingresar a la Escuela Politécnica Nacional en la carrera Tecnología Superior en Desarrollo de Software y está cursando el cuarto semestre. Áreas de interés: Desarrollo WEB, Análisis de datos, Hardware.
(eduardo.farinango@epn.edu.ec)

Marlon Tuquerres, nació en Quito-Ecuador el 10 de octubre de 2000. Realizó sus estudios secundarios en el Colegio Unidad Educativa María Angélica Idrobo. En la actualidad realiza sus estudios en la Escuela Politécnica Nacional en tercer semestre de la carrera de Tecnología Superior en Desarrollo de Software. Áreas de interés: diseño de interfaces, desarrollo web, seguridad informática. (marlon.tuquerres@epn.edu.ec).