



FACULTAD DE INGENIERÍA
TESIS DE MAESTRÍA

Reconocimiento automático de hablantes, empleando técnicas de Deep Learning, en peritajes informáticos.

ALUMNO:

Ing. JUAN MANUEL MIGUEZ

TUTOR DE TESIS:

Mg. MARTÍN ACOSTA

BUENOS AIRES – ARGENTINA 2023

PÁGINA DE APROBACIÓN DEL COMITÉ

Mg. Martín Acosta
Tutor de tesis

JURADO INTERNO 1
Jurado Interno

JURADO INTERNO 1
Jurado Interno

JURADO EXTERNO 1
Jurado Externo

RESUMEN

En el año 1996, investigadores de los laboratorios AT&T Bell (Chin-Hui Lee, Frank K. Soong, Kuldeep K. Paliwal, 1996), definieron “**Reconocimiento Automático de Hablantes**” -ASR por sus siglas en inglés- como “**La ciencia y tecnología orientadas a implementar algoritmos para clasificar e identificar hablantes a través de sus registros de voz**”.

En la actualidad, dicha actividad se encuentra dentro de las incumbencias del perito informático forense. El método científico y el modelo técnico utilizados para llegar a la conclusión, no debe dar lugar a controversias que permitan impugnar el dictamen pericial.

¿Puede un perito en informática, demostrar con convicción que dos registros de voz han sido realizados por la misma persona?

Este trabajo de tesis, contiene un prototipo basado en Inteligencia Artificial (IA) que permite resolver dicha problemática. El experto -en el marco de un proceso judicial- podrá emitir dictámenes sobre la compatibilidad de registros de voz, en contextos acotados, minimizando el riesgo de ser impugnado por motivos técnicos.

Este aporte allana el camino hacia la definición de estándares, metodologías y mejores prácticas relacionados con esta actividad, ya que dicha formalización se encuentra en pleno desarrollo (Dr. Pedro Univaso, 2016)

TABLA DE CONTENIDO

CAPÍTULO 1: INTRODUCCIÓN	17
CAPÍTULO 2: MARCO TEÓRICO	19
Antecedentes	19
Usos y aplicaciones de Reconocimiento de Voz	20
De la voz al comando	20
Dictado automático	20
Control por comandos	20
Sistemas portátiles	21
Sistemas diseñados para discapacitados	21
Del comando a la voz	21
Generador de voz	21
Clonar voces	22
Ataques por clonación de voz (Deep Fake)	22
Educación	22
Audiolibros	22
Tecnología de asistencia	22
Autenticación biométrica por voz	22
Justificación	23
Clasificación de los Sistemas de Identificación de Hablantes	25
Análisis espectrográfico-auditivo	26
Enfoque auditivo-perceptual	26
Enfoque fonético-acústico	27
Métodos automáticos	28
Métodos semi-automáticos	29
Métodos combinados	29
Producción y percepción de las señales de voz	30
Sonido: Armónico simple y complejo	30
Los parámetros de la voz	32
Ancho de banda	32
Armónicos	33
Cavidades resonanciales	33
Cruces por cero (ZC)	33
Energía	33
Espectrograma	33
Formantes	33
Forma de onda	34
Frecuencia	34
Frecuencia fundamental o altura tonal	34
Frecuencias naturales de resonancia	34
Jitter	34
Máximo rendimiento de energía	34
Relación armónico ruido	35
Shimmer	35
Parámetro de referencia en población de prueba	35
Frecuencia Fundamental (F_0)	35

Producción de las señales de voz	36
Fisiología del aparato fonador	38
Cavidades InfraGlóticas	39
Cavidad Glótica	39
Cavidades SupraGlóticas	40
Clasificación de los trastornos de la voz	42
Trastornos orgánicos de la voz	44
Trastornos funcionales de la voz	44
Percepción de las señales de voz	44
Oído Externo	45
Oído Medio	45
Oído Interno	45
Digitalización de la voz	47
Procesos en la conversión Anológico-Digital de la voz	47
Tasa de muestreo	48
Teorema de muestreo de Nyquist-Shannon	49
Aliasing	49
Transformada de Fourier	49
Par de Transformada de Fourier	50
Representación Matlab de señales típicas	51
Transformada Rápida de Fourier	51
Oscilograma, Sonograma y Espectrograma	52
Relación entre IA - ML – DL	54
Inteligencia Artificial	54
Machine Learning	54
Deep Learning	54
Conceptos sobre Deep Learning	57
Limitaciones de Deep Learning	57
Redes Neuronales	57
Modelo biológico de una Red Neuronal	58
Red Neuronal Artificial	62
Modelo de una Red Neuronal Artificial	62
Elementos básicos de una Red Neuronal Artificial	64
Elementos básicos de una Red Neuronal Artificial	70
Clasificación de algoritmos según entrenamiento	71
Algoritmos supervisados	71
Errores Tipo I y Tipo II con ejemplos	73
Todas las métricas	73
¿Qué métrica se ajusta mejor al modelo?	76
¿Cuántas EPOCHS es necesario ejecutar?	77
Relación entre Sensibilidad (Recall), Precision, y Errores Tipo I y II	77
Métricas desde la librería Keras	79
Algoritmos no supervisados	79
CAPÍTULO 3: ARQUITECTURA	80
Arquitectura de Referencia	80
Arquitectura de Sistema	81
CAPÍTULO 4: MARCO METODOLÓGICO	82

Reconocimiento de Voz	82
Primer proyecto	82
Segundo proyecto	84
Tercer proyecto	85
Cuarto proyecto	86
Análisis del Método Automático	87
Librerías utilizadas	88
pytube	88
sox y pysox	89
ffmpeg	90
Pydub	90
Xlrd	90
Ipython.display	91
Librosa	91
Pandas	91
Keras	92
essentia.standard.Monoloader	93
Equipo de procesamiento Local	93
Hardware	93
Software	93
Procesamiento en la nube	94
Conjunto de datos	95
Conjunto preliminar de archivos de voz de hablantes	95
Conjunto de archivos de audio de hablantes sin ruido	96
Conjunto de archivos de voces en idioma español, para pre-procesar	96
Conjunto de archivos de audio para Deep Fake	97
Preparación de ruido	98
Etapa del pre-procesamiento de registros de voz	98
Repositorio - Preparación de datos	99
Pre-procesamiento	100
Preparación de la etapa de pre-procesamiento	100
Función cargarAudio	101
Función convertirMP4aWAV	101
Función identificarArchivoAudioSinPreProcesar	101
Función eliminarArchivoMP4	102
Función normalizarAudioCrudo	102
Función dividirArchivoNormalizadoEnChuncks1seg	103
Secuencia de procesamiento del segundo bloque principal	104
Procesamiento	105
Etapa del procesamiento	105
Configuración inicial	105
Preparación de ruido	106
Generación del conjunto de datos (DataSet)	106
Definición del modelo	107
Entrenamiento	108
Evaluación	108
Demostración	109

Mejoras de desempeño	111
Rendimiento	111
Menor cantidad de hablantes	112
Optimizando VALID_SPLIT de 0,1 a 0,9	113
Resultados	115
Reglas Generales / Restricciones / Interpretación	115
Performance	116
Entrenamiento inicial con 1 voz	117
Imitadores	117
Mismo hablante, pero difieren 20 años.	117
Entrenamiento con set de 5 hablantes (performante)	118
Entrenamiento con hablante y su Deep Fake	118
Entrenamiento con trastorno orgánico de la voz	118
CAPÍTULO 5: CONCLUSIONES	119
Futuras líneas de investigación	121
Someter el modelo a pruebas de certificación y ensayos clínicos.	121
Fase 1	121
Fase 2	121
Fase 3	121
Autenticación de identidad de aplicaciones	121
Diagnóstico automático de afecciones cardiológicas	121
Evolución en el tiempo de la misma persona	121
CAPÍTULO 6: BIBLIOGRAFÍA	122
Citas Bibliográficas	122
Referencias Bibliográficas	123
Documentos Electrónicos	125
CAPÍTULO 7: APÉNDICES	132
Ejecución de pruebas	132
Prueba 1 – entrenamiento con 1 hablante	132
Prueba 2 – entrenamiento con 1 hablante	133
Prueba 3 – entrenamiento con 1 hablante.	134
Prueba 4 – entrenamiento con 1 hablante.	135
Prueba 5 - imitador.	137
Prueba 6 – audio atemporal (voz del pasado)	138
Prueba 7 – hablante de otro sexo sin entrenar el modelo	139
Prueba 8 – Voz “similar” que no se usó para entrenar	139
Prueba 9 - Dos hablantes para entrenar el modelo	140
Prueba 10 - Dos hablantes para entrenar el modelo	142
Prueba 11 - Dos hablantes para entrenar el modelo	142
Prueba 12 - Pruebas de rendimiento - Localhost	143
Prueba 13 – Rendimiento, GPU, 1 Epochs, split 0,1	145
Prueba 14 - Rendimiento, TPU, Epochs 1, split 0,1	145
Prueba 15 - Rendimiento, TPU, 30 Epochs, split 0,3	146
Prueba 16 - Rendimiento, TPU, 10 Epochs, split 0,4	147
Prueba 17 - Rendimiento, TPU, 10 Epochs, split 0,8	149
Prueba 18 – Rendimiento: voces género Femenino	150
Prueba 19 – Rendimiento: voces género Femenino	153

Prueba 20 – Rendimiento: voces género Femenino	153
Prueba 21 – Rendimiento: voces género Femenino	154
Prueba 22 – Rendimiento: voces género Femenino	154
Prueba 23 – Rendimiento: voces género Femenino	154
Prueba 24 – voces género Femenino	155
Prueba 25 – voces género Femenino	157
Prueba 26 – voces género Femenino	157
Prueba 27 – voces género Femenino	158
Prueba 28 – voces género Femenino	158
Prueba 29 – voces género Femenino	159
Prueba 30 – voces género Femenino	159
Prueba 31 – voces género Femenino, split 0.5	160
Prueba 32 – voces género Femenino, split 0.2	162
Prueba 33 – voces género Femenino, split 0.3	164
Prueba 34 – voces género Femenino, split 0.4	166
Prueba 35 – voces género Femenino, split 0.6	168
Prueba 36 – voces género Femenino, split 0.7	170
Prueba 37 – voces género Femenino, split 0.8	171
Prueba 38 – voces género Femenino, split 0.9	173
Prueba 39 – voces género Femenino, split 0.25	175
Prueba 40 – voces género Femenino, split 0.35	177
Prueba 41 – Voz de un imitador vs. Voz real	180
Prueba 42 – Deep Fake	180
Prueba 43 – Voz generada por AI	180
Prueba 44 – Voz generada por AI	184
Prueba 45 – Voz generada por AI, Split 0.5	187
Prueba 46 – Voz generada por AI	189
Prueba 47 – Voz generada por AI	192
Prueba 48 – Voz generada por AI	193
Prueba 49 – Voz generada por AI	193
Prueba 50 – Voz generada por AI	194
Prueba 51 – Voz sin diafonía, con resfrío	195
Problemas encontrados	198
No se puede reproducir audio mpeg4 de YouTube	198
Error con permisos de ejecución en Anaconda.	198
Error de dimensionado de pantalla al abrir Anaconda.	199
Mensajes de advertencia al abrir Anaconda.	199
Cambiar en navegador por defecto de Firefox a Chrome	200
Escasos recursos de memoria y de disco en Google Colab	200
Límite de ejecuciones en Google Colab	200
Método predict, entrenando la muestra de un hablante	201
Código fuente	203
Preprocesamiento	203
Notebook para pre-procesar archivos de voz v0030101	203
Instalar librerías, paquetes y dependencias	203
Importar librerías, paquetes y dependencias	204
Constantes y Variables de sesión	205

Funciones para descargar audio y convertir a *.wav	206
Descargar audio y convertirlo a *.wav	207
Nombre del hablante, Instante de inicio y fin del audio, desde xls	207
Funciones para normalizar Audio	207
Normalización y división en múltiples muestras	208
Procesamiento	209
Notebook para procesar archivos de voz v0070101	209
Instalar librerías, paquetes y dependencias	209
Importar librerías, paquetes y dependencias	210
Variables de sesión	210
Administrar estructuras de directorios para archivos	211
Archivos de audio con ruido	212
Muestreo y división por tiempo de archivos de audio con ruido	212
Funciones para procesar audio	213
Obtener las clases y sus etiquetas	214
Obtener set de datos para entrenamiento y para validación	215
Construir el modelo	216
Entrenar el modelo	217
Imprimir las métricas: Accuracy y Loss	217
Evolución de las métricas: Accuracy y Loss por EPOCHS	218
Matriz de Confusión	218
Verificación del modelo	219
Clasificar con qué categoría es compatible VozViva.wav	220
Dibujar la forma de onda y sonograma de VozViva.wav	221

LISTA DE TABLAS

Tabla 1: Frecuencia promedio de cada vocal.	36
Tabla 2: Comparativo cantidad de operaciones a realizar con FTD vs. FFT.	52
Tabla 3. Métricas. Predicciones vs. Realidad	73
Tabla 4: métricas más comunes.	74
Tabla 5: Resto de métricas usadas en aprendizaje supervisado.	78
Tabla 6: “Métricas Accuracy / Loss” vs. “valid_split / EPOCHS”	180
Tabla 7: Características de HW y SW para pruebas de rendimiento Localhost	144
Tabla 8: Tabla de voces de hablantes de género femenino para Rendimiento.	150
Tabla 9: Tabla de voces de hablantes de género femenino (set reducido)	155
Tabla 10: Hablantes que se procesan para la prueba 43	183
Tabla 11: Hablantes que se procesan en la prueba 44	186
Tabla 12: Hablantes que se procesan en la prueba 46	192
Tabla 13: Hablantes que se procesan en la prueba 51	196

LISTA DE FIGURAS

Ilustración 1: Comparación de 2 sonogramas.	26
Ilustración 2. Formulario auditivo-perceptual para identificación de hablantes.	27
Ilustración 3: Modelo matemático para describir el “movimiento vibratorio”	30
Ilustración 4: Movimiento armónico simple discreto.	31
Ilustración 5: Descomposición de un sonido complejo.	32
Ilustración 6. Elementos que componen el aparato fonador.	38
Ilustración 7. Laringe y repliegues vocales en movimiento.	40
Ilustración 8. Cavidades SupraGlóticas.	41
Ilustración 9. Localización de las frecuencias dentro del caracol.	46
Ilustración 10. Pasos desde que el sonido ingresa al oído, hasta que la información llega al cerebro.	46
Ilustración 11: Pulso rectangular de semianchura 1 y transformada de Fourier.	51
Ilustración 12: Oscilograma para una señal de voz.	52
Ilustración 13: Sonograma para una señal de voz.	53
Ilustración 14: Espectrograma de una señal de voz.	53
Ilustración 15. Diferencias entre ML y DL (pixabay, ID ilustración:1364557896)	55
Ilustración 16. Jerarquías IA , ML y DL (pixabay, ID ilustración:1202474000)	56
Ilustración 17. Neurona biológica (pixabay, ID ilustración:1223014701)	59
Ilustración 18. Cómo la neurona procesa la información y qué salida se obtiene.	59
Ilustración 19. Señal de propagación del impulso eléctrico a lo largo del axón.	61
Ilustración 20. La inversión del voltaje, se propaga por el axón.	61
Ilustración 21. Función escalón	66
Ilustración 22. Función Lineal	67
Ilustración 23. Función Sigmoidal	68
Ilustración 24. Función Gaussiana.	68
Ilustración 25. Modelo supervisado (pixabay, ID ilustración:1310128920)	71
Ilustración 26. Matriz de confusión.	72
Ilustración 27: se puede guardar el mejor modelo si se calcula accuracy.	77
Ilustración 28. Sistema de reconocimiento del hablante4 - Laboratorio de Investigaciones sensoriales CONICET , 2016.	80
Ilustración 157: Arquitectura Sistema de reconocimiento de hablantes utilizando deep-learning.	81
Ilustración 29. Etapas del proceso de pre-procesamiento.	100
Ilustración 30: Detalle del archivo de hablantes con referencia temporal.	101
Ilustración 31: Forma de onda para audio sin preprocessar y audio preprocessado	103
Ilustración 32: Etapas de procesamiento	105
Ilustración 33: Función procesamiento residual con activación relu.	107

Ilustración 34: métricas de una ejecución del prototipo con datos de prueba	109
Ilustración 35: Valor de métrica Accuracy vs. EPOCHS (a la izquierda) y Valor de métrica Loss vs. EPOCHS (a la derecha)	112
Ilustración 36: Grabación de audio con aplicación Audacity.	199
Ilustración 37: Entrenamiento y evaluación, con 1 categoría.	202
Ilustración 38: oscilograma y sonograma de una prueba.	223
Ilustración 39: métricas para la prueba 1.	132
Ilustración 40: predicción para la prueba 1.	133
Ilustración 41: predicción para la prueba 2.	134
Ilustración 42: métricas para la prueba 3.	134
Ilustración 43: predicción para la prueba 3.	135
Ilustración 44: métricas para la prueba 4.	135
Ilustración 45: predicción para la prueba 4.	135
Ilustración 46: métricas para la prueba 5.	137
Ilustración 47: predicción para la prueba 5.	138
Ilustración 48: predicción para la prueba 6.	138
Ilustración 49: predicción para la prueba 7.	139
Ilustración 50: predicción para la prueba 8.	140
Ilustración 51: métricas para la prueba 9.	141
Ilustración 52: predicción para la prueba 9.	141
Ilustración 53: predicción para la prueba 10.	142
Ilustración 54: predicción para la prueba 11.	143
Ilustración 55: métricas para la prueba de rendimiento Localhost	143
Ilustración 56: resultados para la prueba de rendimiento Localhost	144
Ilustración 57: Verificando procesamiento en colab con TPU	146
Ilustración 58: métricas pruebas de rendimiento colab, TPU, 1 EPOCH	146
Ilustración 59: métricas rendimiento colab, TPU, 30 EPOCHs, split 0,3	147
Ilustración 60: métricas rendimiento colab, TPU, 10 EPOCHs, split 0,4	148
Ilustración 61: métricas rendimiento colab, TPU, 10 EPOCHs, split 0,8	149
Ilustración 62: métrica inicial, rendimiento colab, TPU, 5 EPOCHs, split 0,1	150
Ilustración 63: métrica, rendimiento colab, TPU, 5 EPOCHs, split 0,1	150
Ilustración 64: Prueba 18 - recursos consumidos.	151
Ilustración 65: Prueba 18, Métricas / Epochs.	151
Ilustración 66: Métricas para la prueba 18.	152
Ilustración 67: Resultados para la prueba 18.	152
Ilustración 68: predicción para la prueba 18.	153
Ilustración 69: predicción para la prueba 19.	153
Ilustración 70: predicción para la prueba 20.	153

Ilustración 71: predicción para la prueba 21.	154
Ilustración 72: predicción para la prueba 22.	154
Ilustración 73: predicción para la prueba 23.	155
Ilustración 74: Entrenamiento para la prueba 24.	156
Ilustración 75: Métricas para la prueba 24.	156
Ilustración 76: Resultados para la prueba 24.	157
Ilustración 77: predicción para la prueba 24.	157
Ilustración 78: predicción para la prueba 25.	158
Ilustración 79: predicción para la prueba 26.	158
Ilustración 80: predicción para la prueba 27.	159
Ilustración 81: predicción para la prueba 28.	159
Ilustración 82: predicción para la prueba 29.	159
Ilustración 83: predicción para la prueba 30.	160
Ilustración 84: Entrenamiento para la prueba 31.	161
Ilustración 85: Resultados para la prueba 31.	162
Ilustración 86: Métricas para la prueba 32.	163
Ilustración 87: Resultados para la prueba 32.	164
Ilustración 88: Predicción para la prueba 32.	164
Ilustración 89: Métricas para la prueba 33.	165
Ilustración 90: Resultados para la prueba 33.	166
Ilustración 91: Métricas para la prueba 33	166
Ilustración 92: Métricas para la prueba 34.	167
Ilustración 93: Resultados para la prueba 34.	168
Ilustración 94: Métricas para la prueba 34.	168
Ilustración 95: Métricas para la prueba 35.	169
Ilustración 96: Métricas para la prueba 35.	170
Ilustración 97: Predicción para la prueba 35.	170
Ilustración 98: Issue al tomar recursos desde Google colab.	171
Ilustración 99: Métricas para la prueba 36.	171
Ilustración 100: Resultados para la prueba 36.	172
Ilustración 101: Predicción para la prueba 36.	172
Ilustración 102: Métricas para la prueba 37.	173
Ilustración 103: Resultados para la prueba 37.	174
Ilustración 104: Métricas para la prueba 37.	174
Ilustración 105: Métricas para la prueba 38.	175
Ilustración 106: Predicción para la prueba 38.	175
Ilustración 107: Resultados para la prueba 38.	176
Ilustración 108: Métricas para la prueba 39.	177

Ilustración 109: Resultados para la prueba 39.	177
Ilustración 110: Métricas para la prueba 39.	178
Ilustración 111: Métricas para la prueba 40.	179
Ilustración 112: Resultados para la prueba 40.	179
Ilustración 113: Métricas para la prueba 40.	180
Ilustración 114: Preprocesamiento para la prueba 43.	183
Ilustración 115: Métricas para la prueba 43.	184
Ilustración 116: Resultados para la prueba 43.	184
Ilustración 117: Predicción para la prueba 43.	185
Ilustración 118: Preprocesamiento para la prueba 44.	186
Ilustración 119: Métricas para la prueba 44.	187
Ilustración 120: Resultados para la prueba 44.	187
Ilustración 121: Predicciones para la prueba 44.	188
Ilustración 122: Métricas para la prueba 45.	189
Ilustración 123: Resultados para la prueba 45.	190
Ilustración 124: Predicción 1 para la prueba 45.	190
Ilustración 125: Predicción 2 para la prueba 45.	190
Ilustración 126: Preprocesamiento para la prueba 46.	191
Ilustración 127: Métricas para la prueba 46.	192
Ilustración 128: Resultados para la prueba 46.	193
Ilustración 129: Predicción para la prueba 46.	193
Ilustración 130: Predicción para la prueba 47.	194
Ilustración 131: Predicción para la prueba 48.	194
Ilustración 132: Predicción para la prueba 49.	195
Ilustración 133: Predicción para la prueba 50.	195
Ilustración 134: Predicción para la prueba 51.	197
Ilustración 135: Resultados para la prueba 51.	197
Ilustración 136: Predicción para la prueba 51.	198
Ilustración 137: Instalación GStreamer Multimedia Codecs.	199
Ilustración 138: Instalar librerías, paquetes y dependencias.	204
Ilustración 139: Conectar con Google Drive	205
Ilustración 140: Descargar audio y convertirlo a *.wav	208
Ilustración 141: Nombre del hablante, Instante de inicio y fin del audio	208
Ilustración 142: Normalización y división en múltiples muestras.	210
Ilustración 143: montar Drive.	211
Ilustración 144: Instalar librerías, paquetes y dependencias.	211
Ilustración 145: Variables de sesión.	212
Ilustración 146: directorios para archivos.	213

Ilustración 147: Archivos de audio con ruido.	213
Ilustración 148: Muestreo y división por tiempo de archivos de audio con ruido.	214
Ilustración 149: Obtener las clases y sus etiquetas.	216
Ilustración 150: Set de datos para entrenamiento y para validación.	217
Ilustración 151: Construcción del modelo.	218
Ilustración 152: Construcción del modelo.	218
Ilustración 153: métricas: Accuracy y Loss por EPOCHS.	219
Ilustración 154: Matriz de Confusión.	220
Ilustración 155: Verificación del modelo.	221
Ilustración 156: Clasificación.	222

LISTA DE SIGLAS

ADC: Analog to Digital Converter.

AI: Artificial Intelligence (Inteligencia Artificial)

ASR: Automatic Speaker Recognition (Reconocimiento automático hablante)

CFK: Cristina Fernández de Kirchner.

CIV: Fuero Civil en *PJN* -Poder Judicial de la Nación Argentina-

CNN-1D: Red Neuronal Convolucional de 1 Dimensión.

DL: Deep Learning (Aprendizaje profundo)

FA: análisis factorial

FFT: Transformada Rápida de Fourier

GMM: Modelos de mezclas gaussianas (es un paradigma)

GSL: GMM con funciones de núcleo lineales.

HMM: Hidden Markov Models (Modelos Ocultos de Markov)

IAFPA: International Association for Forensic Phonetics and Acoustics.

NAP: proyección de atributos perjudiciales.

PBX: Central de conmutación telefónica Privada Automática.

PLDA: Análisis discriminante lineal probabilístico.

SW: Software.

SVM: Support Vector Machine: clasificador basado en máquinas soporte vectorial que generan súper vectores.

TED: Tecnología, Entretenimiento, Diseño. Relacionado con Charlas TED.

TF: Transformada de Fourier.

TFD: Transformada de Fourier Discreta.

CAPÍTULO 1: INTRODUCCIÓN

La identificación forense de hablantes se encuentra enmarcada dentro de la fonética forense, y más ampliamente, dentro de la lingüística forense (J. Gibbons y M. T. Turell, 2008). Las áreas relacionadas con fonética forense (P. Rose, 2002), son: autenticación de grabaciones, identificación de contenido y la identificación de hablantes – siendo esta última la que será abordada en este trabajo.

Uno de los puntos críticos del problema de identificación forense de hablantes en procesos judiciales es el de identificar cuáles son los modelos de aprendizaje automático -en la actualidad- que pueden ser utilizados, y que éstos presenten bases científicas sólidas sobre el proceso biométrico de reconocimiento de voz, para evitar su impugnación. El resultado debe permitir la elaboración de un dictamen con un alto nivel de certeza como soporte para el proceso judicial.

Dadas las complejidades y nuevas técnicas informáticas para el reconocimiento de hablantes, la actividad técnica dejó de ser un tema exclusivo de la fuerza pública (policía, prefectura, gendarmería), siendo que en la actualidad, está dentro de las incumbencias del perito oficial de la especialidad en Informática (cualquier jurisdicción y fuero)

Actualmente, en temas Judiciales se ha presentado una alta demanda de profesionales que puedan dictaminar con cierto grado de precisión la identificación de un hablante, para lo cual la designación de un experto capaz de elaborar un dictamen de este tenor es crítico (CIV 034762/2017)

El principal obstáculo a resolver es que según el veredicto del caso Daubert (Daubert, 2016), un juez debe realizar “una valoración preliminar de si el

razonamiento o la metodología subyacentes al dictamen son científicamente válidos y si pueden aplicarse apropiadamente a los hechos del caso”. Es decir, se puede rechazar el testimonio de un perito si la metodología científica utilizada por éste no es aceptada por la comunidad científica.

Como referencia dentro del ámbito judicial Argentino, existe un producto desarrollado por el Laboratorio de Investigaciones Sensoriales (INIGEM, CONICET-UBA) denominado: “*projeto BlackVox Forensia*” (Jorge Gurlekian, 2016) que brinda herramientas tecnológicas para aplicaciones de audición y habla, operando de manera privada en este rubro.

A todas luces, un experto en la materia necesita un protocolo que permita desarrollar un proceso para determinar automáticamente si dos registros de voz son compatibles dentro de un intervalo de confianza, para elaborar un informe pericial sustentable, sin necesidad de recurrir a un proveedor cuyo producto podría obligar a responder por tecnicismos que den lugar a controversias.

CAPÍTULO 2: MARCO TEÓRICO

Antecedentes

Los primeros registros históricos sobre el uso de técnicas sistemáticas para la identificación de hablantes se encuentran a partir de los años 40', en la Unión de Repúblicas Socialistas Soviéticas (URSS), en donde se empleaban técnicas de reconocimiento por escucha y discriminación empírica de voces por parte de científicos forenses. En los años 60' se incursionó en la comparación visual de espectrogramas, y tras 10 años de controversias se determinó que el método auditivo era más preciso que el método de comparación visual. Desde los años 70' se cuenta con métodos automáticos que contribuyen al reconocimiento de hablantes para soportar aplicaciones biométricas.

Los científicos forenses de todo el mundo desalentaron históricamente el análisis automático de los registros de audio con fines judiciales en materia de reconocimiento biométrico debido a ciertas propiedades intrínsecas en los registros de audio que dan lugar a controversias técnicas (David L. Faigman, David H. Kaye, Michael J. Saks, y Joseph Sanders, 1997)

A pesar de contar con estas herramientas y sus muchas evoluciones a través de los años y los desarrollos computacionales, no se ha podido consolidar una herramienta dentro del ámbito judicial para dictaminar si un registro de voz en vivo (o voz viva) es compatible con los registros de voz de prueba judicial.

Para realizar la clasificación de voces, dentro del grupo de legítimos investigadores, se deberían incluir a fonetistas, lingüistas, ingenieros y científicos en computación; ya que son quienes poseen conocimientos sobre ciencia de la voz., (H.F.Hollien, 2002)

Actualmente podría empezar a cambiar esa tendencia, apalancada por los procesos de aprendizaje automático que se basan en modelos científicos, expresando resultados cuantitativos en forma de relaciones de verosimilitud (LR); de todas formas, sigue siendo determinante la resolución del juez para validar la identidad del hablante en base a las comparaciones realizadas por los peritos.

Usos y aplicaciones de Reconocimiento de Voz

A parte del marco de estudio de aplicaciones de reconocimiento de hablantes dentro del ámbito jurídico, cabe resaltar que también existen diferentes aplicaciones y usos para el manejo de registros de voz que han ayudado a la evolución y mejora de técnicas científicas junto con tecnologías informáticas. A continuación se presentan las diferentes aplicaciones y usos para el reconocimiento de voz:

De la voz al comando

Dictado automático

Es uno de los usos más comunes en la actualidad. Por ejemplo, el dictado de recetas médicas y diagnósticos o el dictado de textos legales, se usan corpus especiales para incrementar la precisión del sistema.

Control por comandos

Consiste en dar órdenes a un procesador. Por ejemplo, "abrir navegador", "encender luces" o "cerrar ventana". Estos sistemas reconocen un vocabulario reducido, incrementando su rendimiento.

Telefonía

Algunos sistemas PBX permiten a los usuarios ejecutar comandos de voz, en lugar de pulsar tonos. Por ejemplo, el usuario podría decir un número para navegar por un menú.

Sistemas portátiles

Son sistemas de tamaño reducido, como relojes o teléfonos móviles, que tienen restricciones muy concretas de tamaño. Aquí la voz es una solución natural para introducir datos en sus aplicaciones.

Sistemas diseñados para discapacitados

Diseñados para personas con discapacidades, con restricciones para teclear con fluidez, así como para personas con problemas auditivos, que pueden usarlos para obtener texto escrito a partir de habla. Por ejemplo, para que los hipoacúsicos puedan recibir llamadas telefónicas.

Del comando a la voz

Generador de voz

Generadores de contenido de video o audio. Son narraciones virtuales, en lugar de contratar a un locutor profesional, se utilizan herramientas de IA. Se trata de SW que otorga un aspecto realista, natural y de calidad a las voces.

Clonar voces

Se trata de un software (SW) que mediante IA permite clonar la voz de una persona, de manera que luego de entrenar el modelo permite ingresar un texto y escuchar la voz clonada. Existen controversias éticas acerca de la creación de una voz clonada.

Ataques por clonación de voz (Deep Fake)

Para engañar a los sistemas biométricos de voz haciendo parecer que se escuchan reales, mediante la clonación de voz (IA para la simulación de voz sintética) se trata de una nueva generación de estafas de Phishing, o de método de clonación de voz con fines maliciosos.

Educación

Para la enseñanza educativa y los audios en los museos.

Audiolibros

Las voces de las celebridades se pueden usar para narrar libros, y las figuras históricas pueden contar sus propias historias con sus voces.

Tecnología de asistencia

Para ayudar a las personas con discapacidades o con problemas de salud, que afectan su habla.

Autenticación biométrica por voz

Utilizada por bancos, para autenticar el acceso de personas a sus plataformas en línea.

El concepto “Biométrica” se define como: “Reconocimiento Humano”; y en términos técnicos, Biométrica: “Es la técnica automatizada de medir características físicas o rasgos personales de un individuo además de comparar estas características o rasgos con una base de datos, con el propósito de reconocer de qué individuo se trata” (B. Miller, 1997)

En biométrica, las características físicas son “las cosas que somos” y los rasgos personales son “las cosas que hacemos” (R. Chandrasekaran, 1997)

Dentro de las características físicas tenemos:

- Composición química del aroma corporal.
- Características faciales y emisiones térmicas.
- Características de los ojos (retina e iris).
- Huellas digitales.
- Geometría de la mano.

Como los rasgos personales se puede mencionar:

- Firmas.
- Escritura en máquina o sobre papel.
- Voz (habla) - (A. Davis, 1997 y F. James, 1997)

Justificación

En el año 2.010 existían más de 84 mil abogados matriculados en C.A.B.A. (El Diario Judicial, 2010) y según la tendencia, este número venía aumentado año tras

año. Al momento de elaborar este informe, existían 71 mil abogados en ejercicio de actividades (Colegio Público de Abogados de la Capital Federal, 2022)

El listado de peritos informáticos que publicó el Poder Judicial de la Nación Argentina -*al 29 de Mayo de 2022*- cuenta con 38 ingenieros en informática y 76 analistas en informática, para dar soporte a los requerimientos de todas las causas presentadas por los profesionales en leyes antes citados (Poder Judicial de la Nación Argentina, 2022)

Según recientes sondeos (Consejo de Profesionales de Ciencias Informáticas de la Ciudad de Buenos Aires, 2022), la tercera parte de los abogados en ejercicio, mantiene en promedio 3 causas en curso con al menos 1 de ellas requiriendo algún peritaje informático.

Ahora bien, el procedimiento de designación de peritos expertos -en la Ciudad de Buenos Aires- capaces de emitir un dictamen sobre reconocimiento de hablantes, se realiza por sorteo sobre la base de peritos informáticos; demorando en promedio entre 2 semanas a 2 meses la designación del especialista y si el experto no resulta idóneo o se excusa de resolver el caso, se debe repetir el proceso propagando demoras al proceso judicial, citando a modo de ejemplo el caso (CIV 034762/2017)

En el citado caso *ut-supra*, hubo una demora de al menos 2 años en la designación de un perito idóneo, y una vez aceptado el cargo, el experto no logró acceder al expediente para conocer el tenor de la prueba presentada, propagando otra demora de 2 años.

Si el método de resolución de este tipo de problemática se lograra automatizar, entonces el profesional en informática podría agilizar el tiempo de respuesta de sus

asignaciones en tiempo y forma. Si así fuera, el proceso de identificación automática de hablantes no dependería de la subjetividad humana, sino de un proceso estándar con un alto nivel de certeza, que permitiría elaborar informes muy precisos.

Si resultara aplicable este análisis por aprendizaje automático utilizando “deep-learning” (*DL*), todo profesional gozaría de idoneidad para la realización de la tarea;

Como corolario, se aceleraría el proceso judicial debido a que cualquier perito estaría en condiciones de aceptar cargo inmediatamente, sin formar vínculo (o alianza) con el único actor en el mercado que provee servicios de manera privada; otorgando además, beneficios a la comunidad de informáticos que incursionen en dicha actividad.

Como consecuencia de este análisis, una aplicación a futuro es la securización de aplicaciones mediante la integración del reconocimiento biométrico por voz, empleando protocolo OAuth2 con keycloak (Softjourn, 2020), complementando con otra modalidad de autenticación como por ejemplo “reconocimiento biométrico facial”.

Clasificación de los Sistemas de Identificación de Hablantes

A continuación, se enumeran los criterios utilizados para catalogar los métodos de identificación de hablantes.

Análisis espectrográfico-auditivo

Consiste en comparar por medio de espectrogramas y audición, las grabaciones de voz de la prueba y de la voz viva (sospechoso), incluso se pueden considerar otros hablantes para tener puntos de contraste, para así tomar una decisión a partir del examen visual y auditivo, y si corresponden a una misma persona. Dada la subjetividad y poca base científica de esta técnica, “IAFPA” aconsejó no emplearla en casos forenses (International Association for Forensic Phonetics and Acoustics, 2007)

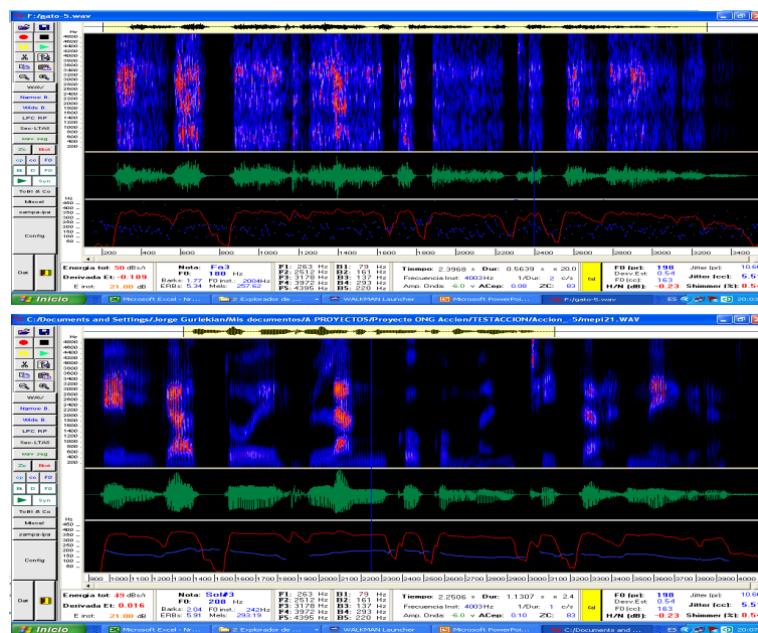


Ilustración 1: Comparación de 2 sonogramas.

Enfoque auditivo-perceptual

Consiste en la participación de un grupo de expertos fonetistas, que empleando sus facultades auditivas y perceptuales realizan la tarea de identificación de un hablante completando una planilla con resultados numéricos.

FORENSIC COMMUNICATION ASSOCIATES		
Case Name:		
<u>Aural-perceptual Approach to Speaker Identification</u>		
<u>Score Sheet -- 0 = U-K least alike; 10 = U-K most alike</u>		
1. PITCH	SCORE	RANGE
a. Level	0 5	10
b. Variability	0 5	10
c. Patterns	0 5	10
2. VOICE QUALITY		
a. General	0 5	10
b. Vocal Fry	0 5	10
c. Other	0 5	10
3. INTENSITY		
a. Variability	0 5	10
4. DIALECT		
a. Regional	0 5	10
b. Foreign	0 5	10
c. Idiolect	0 5	10
5. ARTICULATION		
a. Vowels	0 5	10
b. Consonants	0 5	10
c. Misarticulations	0 5	10
d. Nasality	0 5	10
e.	0 5	10
6. PROSODY		
a. Rate	0 5	10
b. Speech Bursts	0 5	10
c.	0 5	10
7. OTHER		
a. Speech Disorders	0 5	10
b.	0 5	10
MEAN		
fca file:		

Ilustración 2. Formulario auditivo-perceptual para identificación de hablantes.

Enfoque fonético-acústico

Para dictaminar si un registro de voz corresponde a un hablante determinado, este método es ejecutado por un fonetista experto, quien previo a su peritaje requiere de un análisis auditivo, cuyos resultados son las mediciones cuantitativas de parámetros acústicos de las muestras de habla.

El análisis auditivo intenta discriminar las características propias de un oyente para transcribir fonética y fonémicamente (habiéndose recibido de la corte las

transcripciones ortográficas junto a las grabaciones de las voces), determinar propiedades como el tono, calidad de la voz, nivel social, étnico e idioma del hablante; incluso si éste usa métodos de enmascaramiento de la voz, bien sea por medios electrónicos (que pueden llegar a distorsionar voz o género) o no-electrónicos.

Métodos automáticos

El método por modelos de mezclas gaussianas (GMM) se impuso a comienzos de la década del 2000 para las evaluaciones independientes de texto.

La metodología GMM evolucionó hasta la incorporación de clasificadores basados en máquinas de soporte vectorial (SVM – Support Vector Machine) que generan super vectores GMM-SVM con funciones de núcleo lineales (GSL)

Dos nuevos enfoques se han propuesto; el primero basado en el análisis factorial (FA) dentro del paradigma GMM, y el segundo bajo el marco de los clasificadores SVM empleando la proyección de atributos perjudiciales (NAP). Ambos enfoques modelan las variaciones entre múltiples grabaciones de un mismo hablante empleando varios micrófonos y requiriendo el uso de grandes bases de datos, el problema es que el súper-vector generado tiene una gran dimensión.

En el año 2011 se propuso el sistema de reconocimiento de hablantes basado en el análisis factorial como extractor de parámetros. Este análisis define un nuevo espacio de baja dimensionalidad llamado espacio de variabilidad total. En este espacio se representa una emisión del habla por medio de un vector conocido como factor total o i-vector. Para realizar la compensación entre sesiones se propusieron

diferentes técnicas de normalización de vectores, presentando los mejores resultados con el análisis discriminante lineal probabilístico (PLDA)

“Los sistemas basados en i-vectors/PLDA son considerados en la actualidad el estado del arte” (Forensia: un sistema de identificación forense por voz, 2020)

Métodos semi-automáticos

Son aquellos en los que es necesaria la interacción entre el analista y la aplicación de cálculo o análisis que se utiliza.

Grandes laboratorios de acústica forense abordan estrategias diferentes para ejecutar esta metodología. Sin embargo, dado que no hay un estándar de cómo aproximarse al estudio con métodos semiautomáticos, y que estos son empleados según disposición de cada laboratorio no se posee información verídica de cuál ofrece mejor precisión.

Métodos combinados

Los métodos combinados manejan muchas de las metodologías antes mencionadas, como lo son: Espectrográfica-Auditiva, Auditiva-Perceptual, Fonético-Acústica, Semi-Automática y Automática. Entre mayor sea la variabilidad del estudio lo mejor es usar la mayor cantidad de métodos para que los resultados tengan poco margen de subjetividad, además, los métodos combinados son considerados por la comunidad forense como la metodología de mayor fiabilidad.

Producción y percepción de las señales de voz

Sonido: Armónico simple y complejo

Definición de sonido: “Es la sensación producida en el órgano del oído por el movimiento vibratorio de los cuerpos, transmitido por un medio elástico, como el aire”. La vibración es periódica, cuando se repite a intervalos de tiempo determinados.

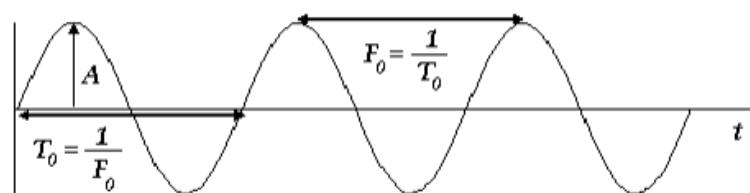


Ilustración 3: Modelo matemático para describir el “movimiento vibratorio”

A : Amplitud de la onda. Representa el valor máximo que puede alcanzar.

T_0 : período. Se expresa en segundos; T_0 es igual a la inversa de la frecuencia: F_0 .

La frecuencia se expresa en ciclos por segundo, hercios o Hertz (Hz)

Cualquier sonido genérico se forma a partir de una suma de movimientos armónicos simples.

Hasta ahora se referenciaron señales en el dominio del tiempo. Sin embargo, una computadora realiza el procesamiento de la información mediante funciones discretas. La diferencia entre una señal discreta y una señal continua, es que la señal discreta no está definida en todo el eje del tiempo, sino que tiene valores únicamente en instantes concretos; que en su mayoría son equidistantes en el tiempo.

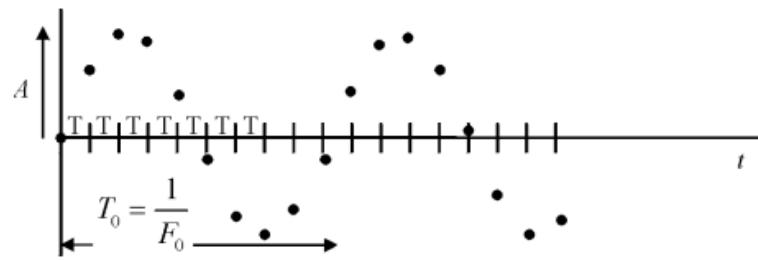


Ilustración 4: Movimiento armónico simple discreto.

T: periodo de muestreo de la onda o el tiempo que se tiene entre dos muestras.

La Frecuencia de muestreo ($F = 1 / T$) se usa para obtener la señal discreta, realizando el cambio de la variable continua t por la variable discreta kT , donde k es la variable discreta.

Donde, de acuerdo al teorema de Nyquist, F debe ser mayor o igual al doble de la frecuencia máxima de la señal.

Ahora podemos definir un nuevo concepto conocido como Sonido Complejo descrito, que está formado por la suma de varios armónicos simples de distinta frecuencia. Para realizar el análisis de este sonido se descompone en los conjuntos de armónicos simples que lo forman, ya que sus intensidades y frecuencias nos informan de las características del sonido original.

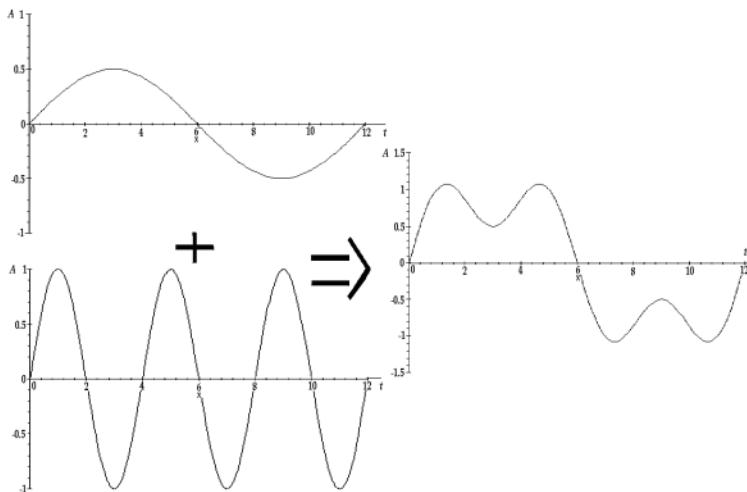


Ilustración 5: Descomposición de un sonido complejo.

La mayoría de los sonidos producidos por el habla son cuasiperiódicos, especialmente las vocales, por lo que para realizar el reconocimiento de las mismas éstas se descomponen en las ondas simples formantes, y de acuerdo a las frecuencias de las mismas se puede distinguir entre las diferentes vocales que existen.

El sonido aperiódico (o ruido) está compuesto por ondas que no presentan ninguna periodicidad. Al descomponer el sonido en sus armónicos básicos, resulta que la energía está muy dispersa en el conjunto de armónicos posibles y resulta poco estable a lo largo del tiempo.

Los parámetros de la voz

Los principales parámetros de la voz son:

Ancho de banda

Valor en Hertz, correspondiente a la diferencia de frecuencias que se obtiene al descender 3 decibeles (dB) del valor máximo del pico del espectro.

Armónicos

En una señal periódica, son las frecuencias múltiplo de la frecuencia fundamental. Algunos armónicos pueden coincidir con las frecuencias de los formantes.

Cavidades resonanciales

Concepto físico, sobre segmentos volumétricos en que puede dividirse el tracto por encima de la glotis hasta los labios. Puede clasificarse en cavidad anterior, media y posterior para modelos de 3 tubos o cavidad faríngea, constricción, cavidad oral. Existen también las cavidades, nasal, faríngea y extra, entre otras.

Cruces por cero (ZC)

Mide los contenidos de frecuencias presentes en la señal. Se basa en la cantidad de veces que la señal cruza el nivel cero de referencia, se cuantifica el cambio de signo en las muestras de la señal digitalizada.

Energía

Representación de la onda basada en la sumatoria al cuadrado de los valores de la amplitud de todas las frecuencias durante un corto tiempo. Se mide en decibeles (dB) y caracteriza la sonoridad.

Espectrograma

Visualización del espectro conformado por la representación de las energías (eje y) para cada componente de frecuencia (eje x) para un segmento de onda de corta duración de 2 a 3 ciclos de frecuencia fundamental (F_0)

Formantes

Frecuencias naturales de resonancia de todas las cavidades supraglóticas, en el momento de producir un sonido determinado. F_1 , abertura de la mandíbula; F_2 ,

forma del cuerpo de la lengua; F3, posición de la punta de la lengua; F4, F5, formantes del cantante. Estos caracterizan el timbre.

Forma de onda

Representación de una señal acústica continuamente variable en amplitud y frecuencia en el tiempo.

Frecuencia

Es la medida de rapidez de la oscilación de una onda por unidad de tiempo. Indica la cantidad de veces que un ciclo de la señal se repite en un segundo.

Frecuencia fundamental o altura tonal

Corresponde a la vibración de los pliegues vocales, se mide en Hz, y cuantifica el número de aperturas y cierres de la glotis por segundo. Este aspecto no tiene diferencias entre personas con o sin educación vocal, lo que hace la diferencia es de qué manera se hacen la fase de apertura y cierre.

Frecuencias naturales de resonancia

Frecuencias de la máxima energía de una cavidad resonancial. Estas frecuencias varían constantemente de acuerdo a la posición de los órganos móviles.

Jitter

Mide la variación del período del tono fundamental ciclo a ciclo. Se basa en la detección de picos periódicos máximos en la señal y en la premediación de la desviación en cada ciclo. Se mide en mili-segundos.

Máximo rendimiento de energía

Se produce cuando la frecuencia del armónico y la frecuencia del formante se aproximan hasta igualarse. En esta condición la energía del formante se eleva y disminuye el ancho de banda.

Relación armónico ruido

Mide la relación de energía en dB que tiene un componente periódico de la señal con respecto a los componentes ruidosos. Si el Jitter es alto, la medida no es válida (Yumoto Et Al, 1982)

Shimmer

Variación de la amplitud de la onda ciclo a ciclo. Se basa en el grado de correlación que tienen los picos periódicos de la función de auto-correlación.

Una correcta determinación de la posición y evolución de los formantes del habla ayudaría enormemente a avanzar en el reconocimiento y síntesis de la voz (Jesús Bobadilla, Pedro González y Jesús Bernal, 1999)

Parámetro de referencia en población de prueba

Los valores de referencia obtenidos para la caracterización de los indicadores acústicos de la voz normal en la población urbana de una ciudad hispano-latina (Libia María Botero Tobón, 2008), cuya edad promedio fue de 20 años para el género femenino y de 22 años para el género masculino; los resultados son:

Frecuencia Fundamental (F_0)

En Latinoamérica, la F_0 en el género femenino oscila -en promedio- entre 226 y 237 Hz, en ascenso desde la más grave vocal /u/, hasta lograr en la /i/ una diferencia de 10 Hz. Este parámetro se ubica en la escala de La3=440 Hz en la2 y la#2, según la escala suramericana de Jackson y en la Anglosajona se ubica en la3 – la#3.

En el género masculino la F_0 -en promedio- presenta un rango de 127 a 132 Hz, ascendiendo en 1 Hz en cada vocal, en este caso la /u/ fue la más aguda. Se ubica en

la escala de La3=440 Hz en Do2 según escala suramericana de Jackson y en la anglosajona en un Do3.

GENERO	VOCALES				
	A	E	I	O	U
FEMENINO	227,8	228	237	234,8	226,3
MASCULINO	127	128	129	130,7	131,8

Tabla 1: Frecuencia promedio de cada vocal.

Los resultados son compatibles con el Análisis acústico de la voz normal y patológica utilizando dos sistemas diferentes (Anagraf y Praat, 2012). Ambos sistemas son programas informáticos de uso común en Latinoamérica, en contextos clínicos y de investigación, para detectar y caracterizar el habla, la voz y los desórdenes vocales.

Sobre un total de 776 muestras de voz correspondientes a 4 repeticiones de la vocal /a/ de 194 hablantes de español en Buenos Aires se midieron utilizando los parámetros disponibles como lo son: la frecuencia fundamental, jitter, shimmer y harmonic-to-noise ratio. Los resultados muestran valores similares de frecuencia fundamental (F_0) para ambos programas.

Dichas referencias, son utilizadas más adelante para configurar las variables del proyecto y asegurar resultados confiables.

Producción de las señales de voz

La forma en la que el ser humano crea las señales de voz, se resume en cuatro pasos de la siguiente manera:

1. El diafragma empuja los pulmones, haciendo que se expulse el aire.

2. El aire circula por la traquea y laringe, pasando por las cuerdas vocales y haciendo que vibren con un tono fundamental.

3. El tono fundamental producido por las cuerdas vocales pasa a través de la laringe, a la caja de resonancia que forman las cavidades nasales y oral.

4. Algunas frecuencias entran en resonancia en las cavidades nasales y oral, saliendo hacia el exterior como la información más importante del habla.

La producción de las señales de voz, se realiza a tres niveles, los cuales se consideran para la creación de sistemas de reconocimiento:

- Nivel Fonológico. Estudia las unidades lingüísticas mínimas que son los Fonemas. Éstos se establecen por posición, si se cambia un sonido de una palabra, el significado de la palabra cambia totalmente.

- Nivel Morfosintáctico. Estudian las palabras estableciendo su género, número y tiempo.

- Nivel Semántico. Se estudia el significado de las frases y su coherencia.

En la mayoría de los sistemas de reconocimiento de hablantes, el análisis se basa en el nivel fonológico, tomando en cuenta que lo podemos dividir en dos vertientes: Fonética articulatoria y Fonética Acústica.

La fonética articulatoria estudia el movimiento de los órganos fonadores para la formación y emisión del sonido.

La fonética acústica estudia las características de la onda sonora y su percepción, la cual es más utilizada en los sistemas de reconocimiento de hablantes.

Fisiología del aparato fonador

La voz humana se describe como una función secundaria del aparato respiratorio, y aunque la necesidad de respirar es constante y vital, ello no quita la importancia al papel jugado por el aparato respiratorio/fonador en la comunicación oral entre las personas.

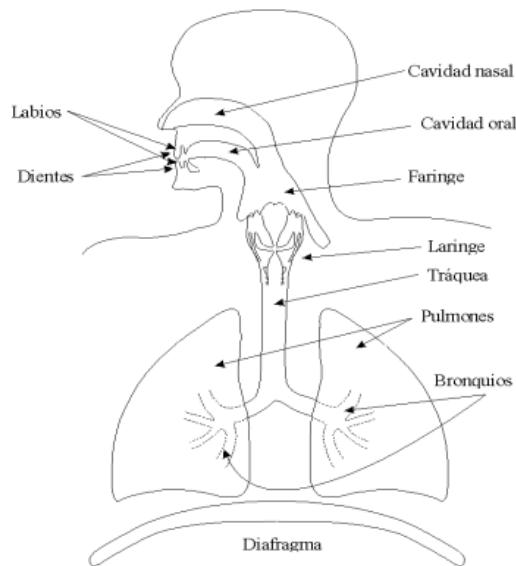


Ilustración 6. Elementos que componen el aparato fonador.

El aparato fonador se puede dividir en tres partes importantes:

- Cavidades Infraglóticas.
- Cavidad Glótica.
- Cavidades Supraglóticas.

Cada una de estas partes realizan una función distinta durante la fonación, pero todas ellas son imprescindibles en la misma.

Cavidades InfraGlóticas

Proporcionan la corriente de aire espirada necesaria para producir el sonido. Las componen el diafragma, pulmones, bronquios y tráquea.

El diafragma es un músculo situado por debajo de los pulmones y con forma de cúpula. Éste controla el despliegue e hinchado de la cavidad pulmonar o su reducción y vaciado junto con los músculos pectorales, y con ello la respiración.

En el momento en el que el diafragma se contrae, la cavidad torácica se ensancha, produciendo la inspiración del aire; al relajarse la cavidad se reduce produciendo entonces la espiración del aire contenido en los pulmones.

Los bronquios y la tráquea son tubos cartilaginosos que conducen el aire entre los pulmones y la laringe. Su función en la fonación es la de simples canales de transmisión del flujo aéreo.

Cavidad Glótica

Está formada por la laringe. Contiene las cuerdas vocales, que son las responsables de la producción de la vibración básica para generar la voz. Aunque tradicionalmente se llaman cuerdas vocales, en realidad se trata de dos marcados pliegues musculosos, cuando el aire sale de los pulmones y pasa por la ranura glótica (la glotis es el espacio triangular que queda entre las cuerdas vocales), las hace vibrar. La vibración producida puede variar en frecuencia e intensidad según varíe la masa, longitud y tensión de las cuerdas vocales.



Ilustración 7. Laringe y repliegues vocales en movimiento

En la Figura, se muestra un corte transversal del elemento principal de la cavidad glótica: la laringe. Donde también se muestran los movimientos de los repliegues vocales, indicados por las líneas continuas y discontinuas. Las flechas indican el movimiento hacia donde se lleva acabo el movimiento de esta cavidad.

Cavidades SupraGlóticas

Está formada por cuatro cavidades: faríngea, nasal, bucal y labial.

Arriba de la laringe, se encuentra la faringe y es ahí donde inicia la raíz de la lengua. Después está el primer obstáculo móvil: la úvula; es el apéndice final del paladar blando o velo del paladar. Cuando está unida a la pared faríngea, la corriente de aire sale exclusivamente por la boca, produciendo sonidos orales. Si el velo del paladar está caído, también se expulsará aire por la cavidad nasal.

La cavidad nasal carece de elementos móviles, por lo que su función es pasiva en la producción del habla. La lengua es el órgano más móvil de la boca, registra una actividad elevada durante el habla. Se divide en tres partes: raíz, dorso y ápice. Estudios recientes han demostrado que el perfil que adopta cada movimiento es causa de un resonado acústico y entonces el timbre será diferente.

Según la forma que tenga podrá ser: cóncava, convexa o plana; o si se sitúa en la parte anterior, central o posterior.

Dentro de la cavidad bucal se encuentran los dientes y los alvéolos.

Los dientes son órganos pasivos, dependiendo en donde se encuentren insertados. Los que se encuentran en la mandíbula inferior son pasivos en muy poca medida ya que la mandíbula inferior es la única activa durante la articulación. Los que se encuentran en la mandíbula superior son completamente pasivos.

El paladar es una amplia zona que va desde los alvéolos hasta la úvula. En ella se distingue el paladar duro, situado sobre el hueso palatino, y el paladar blando o velo del paladar que acaba en la úvula. Los labios son el elemento que posee mayor movilidad y por lo tanto son los encargados de crear y modificar los sonidos.

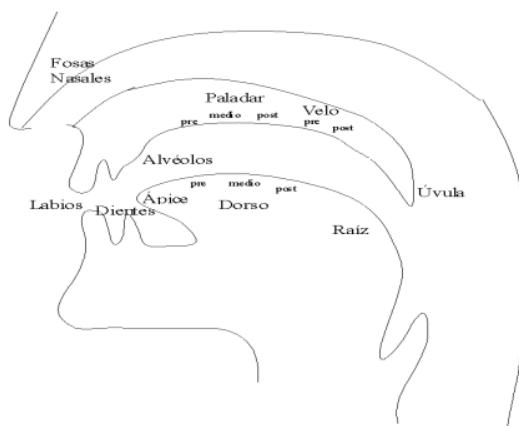


Ilustración 8. Cavidades SupraGlóticas.

Para explicar la generación de las señales de voz, se debe contar con los siguientes elementos:

- Una fuente de energía, proporcionada por el aire a presión que se expulsa en la inspiración.
- Un órgano vibratorio: las cuerdas vocales.

- Una caja de resonancia: las fosas nasales, la cavidad bucal y la faringe.
- Un sistema de articulación del sonido; lengua, labios dientes y úvula.

Frecuencia del Fundamental

El proceso de producción se inicia con la expiración del aire, que al pasar a través de las cuerdas vocales las hace vibrar a una frecuencia determinada que depende de la tensión de las mismas; a esta frecuencia se le conoce como frecuencia del fundamental. El tono está relacionado con la frecuencia del fundamental; cuando el tono es grave indica que la frecuencia es baja y cuando es agudo que la frecuencia es alta. Del modo en que se encuentren articulados los órganos se formará una caja de resonancia distinta, la cual potenciará un conjunto de frecuencias y atenuará el resto.

Aunque articulemos de forma similar los distintos fonemas, según la distancia, forma, dureza, etc. de los órganos, aparecen características especiales de cada individuo; que para nuestro estudio es la información y característica más importante para lograr una eficiencia aceptable en el funcionamiento del sistema de reconocimiento de hablantes.

Finalmente la voz sale al exterior, por lo tanto este proceso explica el conjunto de fonemas sonoros, los demás se producen por fricciones y explosiones de aire.

Clasificación de los trastornos de la voz

En el marco interdisciplinario dominado por la lingüística, la voz es una expresión de la intervención de sistemas orgánicos y de su interacción con el medio ambiente y la cultura.

Los problemas vocales pueden ir desde leves hasta severos. Es necesario detectar “vicios” vocales que podrían impedir el éxito del dictamen (más adelante se detallan los resultados)

El uso inadecuado y el abuso vocal, podría desencadenar características vocales como: voz soplada, tensa, estrangulada, voz con quiebres, afonía (que es la pérdida total del sonido) y disfonía. Es importante remarcar que un trastorno de la voz podría desencadenar en el pedido de nulidad de la prueba o la impugnación del informe pericial si pasa desapercibido.

La **ronquera** es una alteración patológica del timbre de la voz que aparece exclusivamente por alteraciones anatómicas, debido a irregularidades de los bordes libres de las cuerdas vocales, por irregularidad de las oscilaciones de las mismas o por cierre insuficiente de la hendidura glótica, que se caracteriza en el cuadro acústico por aperiodicidad de la frecuencia fundamental y de los sobre-tonos armónicos respectivamente, a través de ruido turbulento; dicho cuadro acústico se puede percibir y medir (Augspach, F. 2003)

La **fono-astenia**, está definida como cansancio en la voz de diferente severidad, modificación del timbre posterior al uso vocal, reducción en la extensión y la tesitura, esfuerzo en la emisión, dificultad en el manejo de la intensidad en especial la suave, disminución en la flexibilidad sonora, en el manejo de curvas melódicas, in-coordinación fono-respiratoria, corta duración en la emisión.

Los principales trastornos de la voz se clasifican según:

Trastornos orgánicos de la voz

Se asocian a la presencia de alguna patología evidente. Los trastornos vocales funcionales son alteraciones en las cualidades de la voz sin lesión aparente en los sistemas que intervienen en la emisión vocal, pero cuando persisten dichos síntomas en general por una técnica vocal inapropiada se puede dar inicio a compensaciones que desencadenan el hábito hiperfuncional convirtiendo esta en causas con compromiso orgánico como son las denominadas mixtas.

Trastornos funcionales de la voz

Han sido llamados disfonías por tensión muscular, porque la laringoscopia usualmente muestra patrones anormales de biomecanismos laríngeos (Molina, N., 2002). El funcionamiento vocal básico se define como la emisión de una voz con diferentes características acústicas en intensidad, tono y timbre con suficiente facilidad y resistencia a la fatiga, por ello cualquier alteración pueden determinar limitaciones de la funcionalidad vocal y la inteligibilidad (Acoustic Measures and Self-reports of Vocal Fatigue by Female Teachers, 2006)

Percepción de las señales de voz

La función principal del oído es captar las ondas acústicas y transformarlas en impulsos nerviosos que el cerebro pueda interpretar. La manera en que trabaja es descomponiendo las señales auditivas que le llegan en sus frecuencias fundamentales.

El oído está formado por tres partes importantes, que son las encargadas de realizar la descomposición de las señales que éste capta:

- Oído externo.
- Oído medio.

- Oído interno.

Oído Externo

Está formado por el pabellón auditivo y por el conducto auditivo externo. El pabellón auditivo (conocido como oreja) recoge las ondas sonoras y facilita su paso hacia el interior, realizando una amplificación de las mismas. El conducto auditivo externo acaba en el tímpano, que es una membrana que lo separa del oído medio. Este conducto lleva las ondas hacia la membrana y las amplifica, pero atenúa aquellos sonidos más agudos que podrían dañar a la cóclea.

Oído Medio

Se separa del oído externo a través del tímpano y se comunica con el oído interno a través de la ventana oval y la ventana redonda. Dispone de una cadena ósea de tres pequeños huesos: martillo, yunque y estribo. Se encuentran en él la trompa de Eustaquio, un canal que se comunica con faringe; la misión del oído medio es transmitir los sonidos desde el oído externo al interno realizando una adaptación de la impedancias acústicas. Cuando la intensidad es pequeña, la cadena ósea se mueve en conjunto produciendo un aumento de la misma; cuando la intensidad es grande, se produce una disminución de la intensidad para evitar daños al oído interno.

Oído Interno

Está formado por el caracol y el órgano vestibular. El caracol es el órgano de audición y su función es percibir las frecuencias de las vibraciones sonoras, éstas las convierte en impulsos nerviosos que son transmitidos al cerebro para su interpretación.

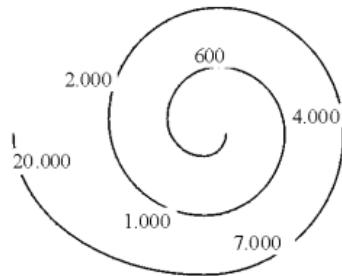


Ilustración 9. Localización de las frecuencias dentro del caracol.

El órgano vestibular está formado por canales semicirculares que intervienen en el equilibrio del ser humano. La percepción de los sonidos es la forma en que cada individuo siente los diferentes sonidos, ésta es una cuestión completamente subjetiva.

El oído humano es capaz de percibir un rango de frecuencias que van desde los 20 Hz hasta los 20,000 Hz, aunque estos límites dependen de la persona en cuestión.

El conjunto de frecuencias y la sensación de diferencia entre las mismas no se percibe con la misma sensibilidad, las frecuencias altas y las bajas se escuchan con menor intensidad, siendo la zona de 3000 Hz la mayor fuerza. La sensación de variación de frecuencias sigue una escala logarítmica, para notar la misma diferencia entre varias frecuencias debemos ir duplicando su valor.

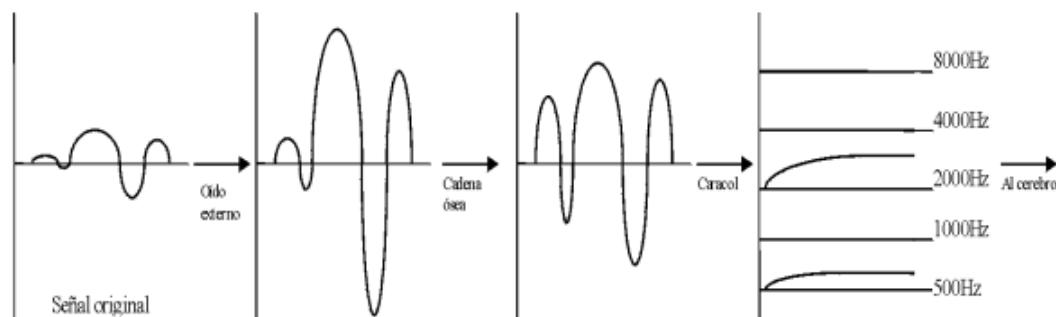


Ilustración 10. Pasos desde que el sonido ingresa al oído, hasta que la información llega al cerebro.

Primero se simplifica el sonido y posteriormente se equilibran los diferentes volúmenes; el sonido se transforma en su representación a través del caracol, del cual salen un conjunto de señales nerviosas que son las que van al cerebro.

Digitalización de la voz

Digitalización es la acción de convertir en digital información analógica. En otras palabras, es convertir cualquier señal de entrada continua (analógica) en una serie de valores numéricos. Este proceso se llama muestreo o sampling.

La señal digital, es la codificación de la señal sonora en términos discretos (en nuestro caso, en dos estados: 0 ó 1).

Procesos en la conversión Anológico-Digital de la voz

En la transformación de la señal de voz desde el sistema analógico hasta el sistema digital, existen dos procesos principales:

Muestreo

(capturas instantáneas obtenidas a partir de la señal analógica) Consiste en tomar muestras periódicas de la amplitud de onda. La velocidad con que se toma esta muestra, es decir, el número de muestras por segundo, es lo que se conoce como frecuencia de muestreo.

Retención

Las muestras tomadas han de ser retenidas (retención) por un circuito de retención (hold) el tiempo suficiente para permitir evaluar su nivel (cuantificación) Desde el punto de vista matemático este proceso no se contempla, ya que se trata de un recurso técnico debido a limitaciones prácticas, y carece, por tanto, de modelo matemático.

Cuantificación

En el proceso de cuantificación se mide el nivel de voltaje de cada una de las muestras. Consiste en asignar un margen de valor de una señal analizada a un único nivel de salida. Incluso en su versión ideal, añade, como resultado, una señal indeseada a la señal de entrada: el ruido de cuantificación.

Codificación

(el valor que representa cada una de esas capturas; a mayor número de bits utilizados para representar el valor, mayor parecido con la señal analógica)

La codificación consiste en traducir los valores obtenidos durante la cuantificación al código binario. Hay que tener presente que el código binario es el más utilizado, pero también existen otros tipos de códigos que también son utilizados.

Tasa de muestreo

La tasa o frecuencia de muestreo, es la cantidad de muestras por unidad de tiempo que se obtienen a partir de una señal continua para producir una señal discreta, durante el proceso necesario para convertirla de analógica en digital.

Las frecuencias se expresan en hertz (Hz o ciclos por segundo) o múltiplos suyos, como el kilo Hertz (kHz), aunque pueden utilizarse otras magnitudes.

En audio, frecuencia máxima perceptible por el oído humano está en torno a los 20 kHz, por lo que una frecuencia de muestreo de 40 kHz sería adecuada para digitalizarla; no obstante, el estándar introducido por un CD-Audio (Disco Compacto), se estableció en 44,1 kHz.

Teorema de muestreo de Nyquist-Shannon

El Teorema de Muestreo de Nyquist explica la relación entre la velocidad de muestreo y la frecuencia de la señal medida. Afirma que la velocidad de muestreo f_s debe ser mayor que el doble del componente de interés de frecuencia más alto en la señal medida. Esta frecuencia por lo general se conoce como la frecuencia Nyquist, f_N .

$$f_s > 2 * f_N$$

La velocidad de muestreo debe ser mayor al doble de la frecuencia Nyquist.

En resumen, el teorema demuestra que toda la información de una señal contenida en el intervalo temporal entre dos muestras cualesquiera, está descrita por la serie total de muestras siempre que la señal registrada sea de naturaleza periódica (como lo es el sonido) y no tenga componentes de frecuencia igual o superior a la mitad de la tasa de muestreo; no es necesario inventar la evolución de la señal entre muestras.

Aliasing

Si una señal es muestreada a una velocidad de muestreo menor que el doble de la frecuencia Nyquist, aparecen componentes de frecuencias bajas falsas en los datos muestreados. Este fenómeno se conoce como aliasing.

Transformada de Fourier

Mediante la Serie de Fourier, podemos representar una señal periódica en términos de sus componentes sinusoidales, cada componente con una frecuencia en particular. La Transformada de Fourier permite hacer esto mismo con señales no periódicas (*Transformada de Fourier - Matlab, 2016*)

Sea f una función real definida en el dominio continuo, dígase $f_{(t)}$ definida en el dominio t . Entonces, la Transformada de Fourier ($F_{(t)}$) se define como:

$$\mathcal{F}(f_{(t)}) = F(\omega) = \int_{-\infty}^{+\infty} f_{(t)} e^{-j\omega t} dt, \quad \forall -\infty < \omega < +\infty \quad (1)$$

Se dice que una señal $f_{(t)}$ tiene Transformada de Fourier si la integral de la ecuación (1) converge (es decir, existe). La integral converge si $f_{(t)}$ “se comporta bien” y si es completamente integrable; esta última condición significa que:

$$\int_{-\infty}^{+\infty} |f_{(t)}| dt < \infty \quad (2)$$

La señal constante no converge, por lo tanto no tiene TF (Transformada de Fourier). Tal es el ejemplo como se muestra a continuación:

$$f_{(t)} = 1, \quad \forall -\infty < t < +\infty \quad (3)$$

Par de Transformada de Fourier

$$f_{(t)} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{j\omega t} d\omega \rightarrow \text{Antitransformada de Fourier}$$

$$F(\omega) = \int_{-\infty}^{+\infty} f_{(t)} e^{-j\omega t} dt \rightarrow \text{Transformada de Fourier}$$

Para que exista la **TF** de $f_{(t)}$, se debe cumplir que:

$$\int_{-\infty}^{+\infty} |f_{(t)}| dt < \infty$$

Representación Matlab de señales típicas

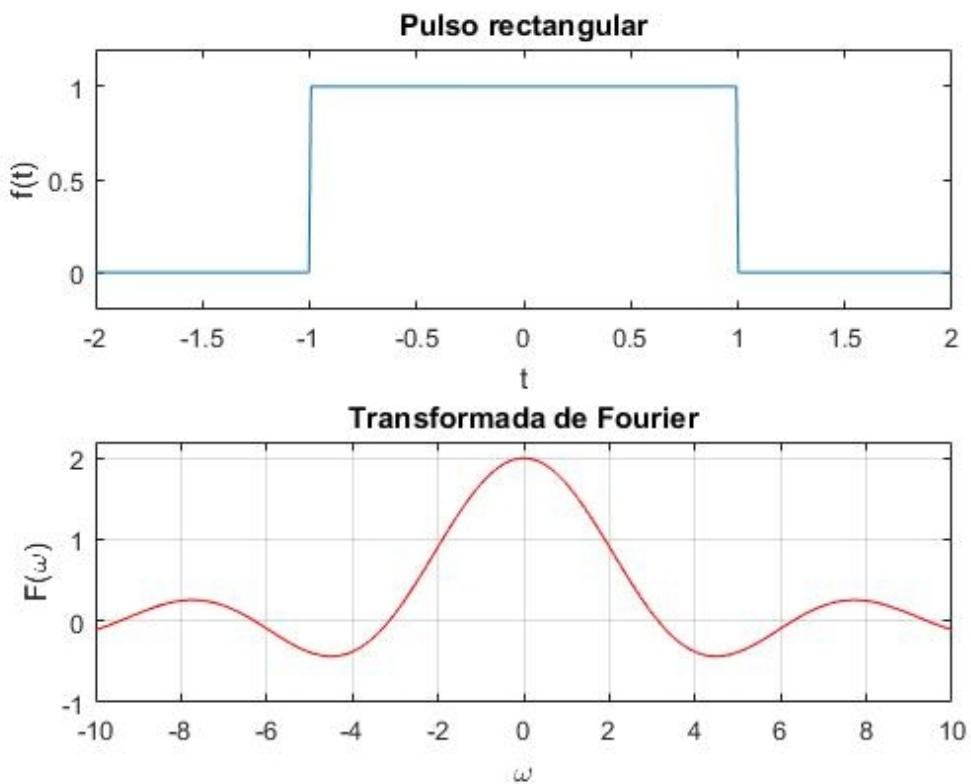


Ilustración 11: Pulso rectangular de semianchura $a=1$ y su transformada de Fourier

Transformada Rápida de Fourier

La Transformada Rápida de Fourier es un algoritmo que permite calcular eficientemente la Transformada de Fourier Discreta y su inversa (Schmidt, Ana Lucía, 2013)

La Transformada Rápida de Fourier está basada en la división del tiempo, eliminando así gran parte de los cálculos repetitivos que hay que llevar a cabo si se desea resolver la TFD de forma directa. Si hacemos una comparación del costo de los dos métodos, el cálculo directo de la TFD y la FFT, podemos observar el factor de mejora que brinda la FFT.

Consideremos el ejemplo presentado, para el tiempo para $N=2^{30}$, donde el cálculo total es de 13343 días. Ahora, aplicando la FFT resulta que hay que realizar

sólo 30×2^{30} operaciones; asumiendo nuevamente que cada operación tarda 1ns, tenemos que el tiempo de cálculo total es de aproximadamente 32 segundos.

<i>N</i>	<i>Nº de operaciones usando cálculo directo (N^2)</i>	<i>Nº de operaciones usando FFT ($N \cdot \log_2 N$)</i>	<i>Factor de Mejora</i>
4	8	4	2,0
8	64	12	5,3
16	256	32	8,0
32	1.024	80	12,8
64	4.096	192	21,3
28	16.384	448	36,6
256	65.536	1.024	64,0
512	262.144	2.304	113,8
1.024	1.048.576	5.120	204,8
2^{30}	2^{60}	30×2^{30}	35.791.394,1

Tabla 2: Cuadro comparativo sobre la cantidad de operaciones a realizar con FTD vs. FFT.

Oscilograma, Sonograma y Espectrograma

Oscilograma

Representa la amplitud de la señal de audio en función del tiempo.

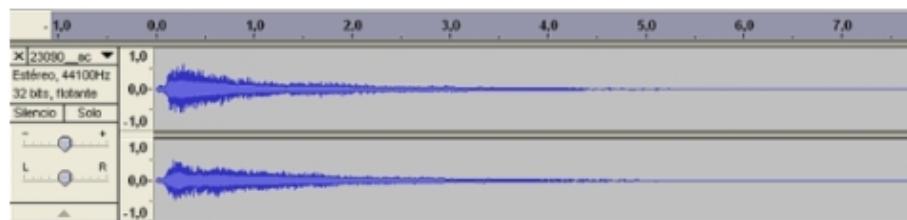


Ilustración 12: Oscilograma para una señal de voz.

Sonograma

Representada por la intensidad de frecuencia en función del tiempo.

El eje vertical corresponde a la frecuencia en kilo Hertz. El eje horizontal al tiempo en segundos. Y el brillo, (más claro o más oscuro) de cada punto de la imagen, corresponde a la intensidad del sonido de cada frecuencia en cada uno de los instantes.

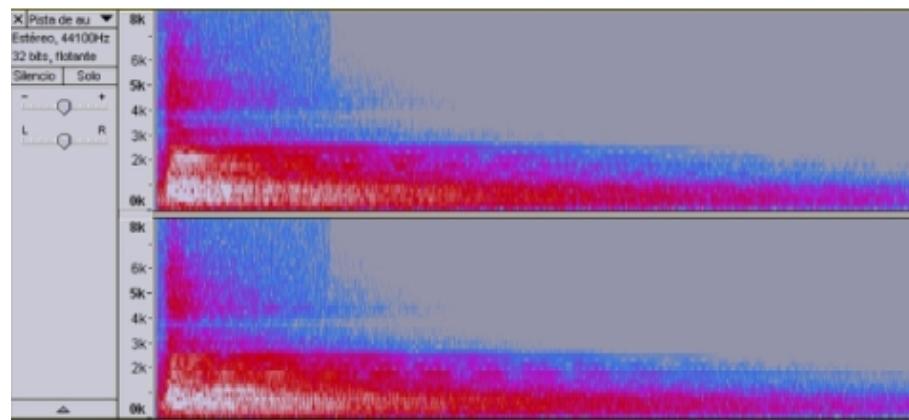


Ilustración 13: Sonograma para una señal de voz.

Espectrograma

Representado por la amplitud de frecuencias armónicas por instante de tiempo.

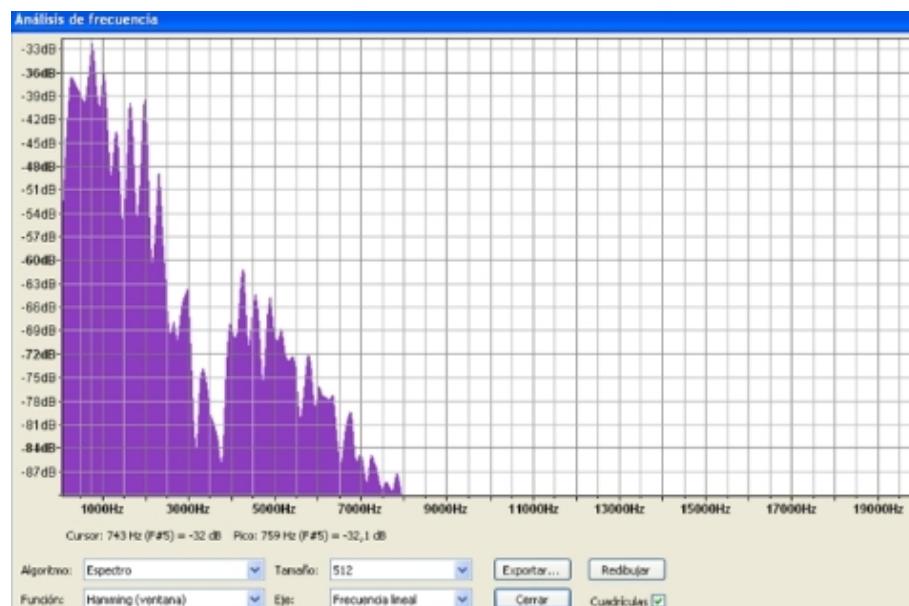


Ilustración 14: Espectrograma de una señal de voz.

Relación entre IA - ML – DL

Inteligencia Artificial

La inteligencia artificial intenta comprender las entidades inteligentes (humanos), con el propósito de construir entidades inteligentes además de entenderlas.

Relacionada con la producción de máquinas para automatizar tareas que requieren comportamiento inteligente.

Machine Learning

Machine Learning es un subconjunto de IA. Se trata de Computadoras con la habilidad de aprender sin ser explícitamente programados.

Recibe un conjunto de datos y aprende por sí mismo.

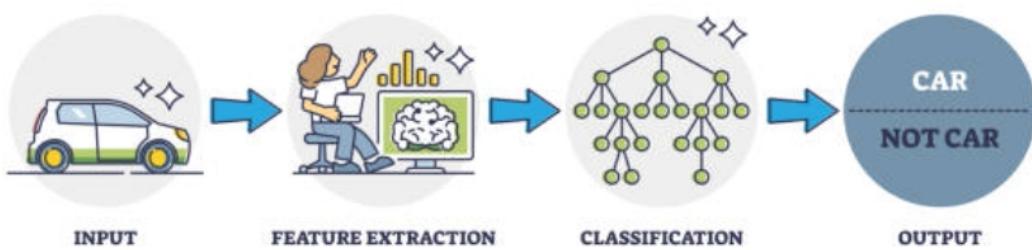
Reconoce patrones entre los datos y hace una predicción.

No requiere que una persona programe instrucciones.

Deep Learning

Es parte del campo de Machine Learning que aprende representaciones de datos (reglas, relaciones, estructuras)

MACHINE LEARNING



DEEP LEARNING



Ilustración 15. Diferencias entre ML y DL

Utiliza algoritmos de aprendizaje que derivan conocimiento a partir de datos, usando jerarquía multível, imitando el cerebro humano.

A partir de datos sin procesar, se emplean patrones para aprender un modelo que luego se usa para clasificar o hacer regresión.

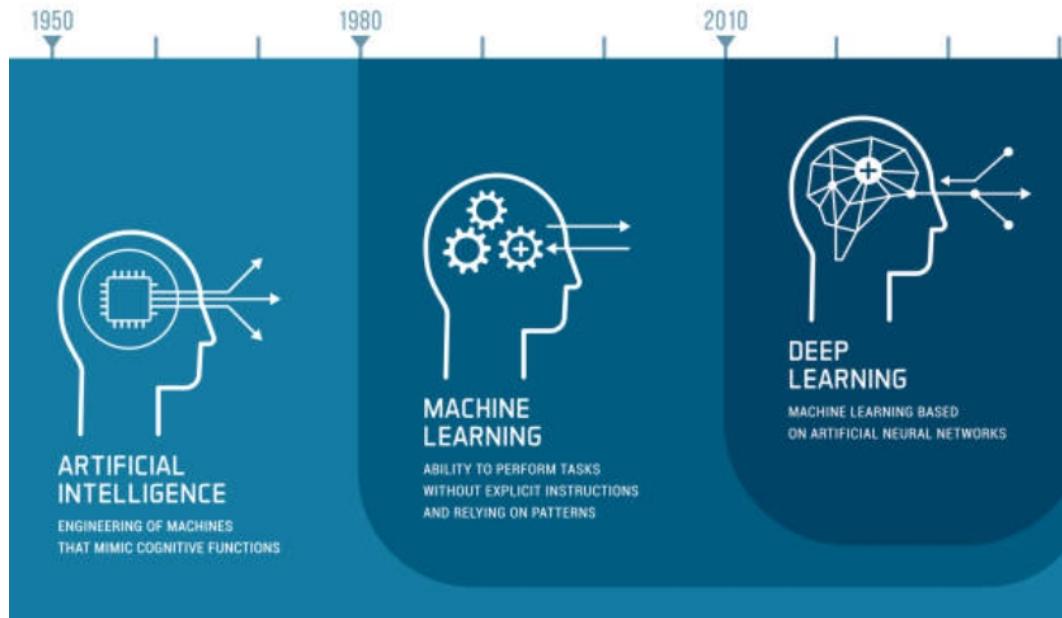


Ilustración 16. Jerarquías IA , ML y DL.

Se puede hacer Deep Learning (DL) para algoritmos supervisados, no supervisados o por aprendizaje por refuerzo. En el caso de DL, se puede manejar abstracción de datos, cuando con las técnicas tradicionales no se podría encontrar un patrón (en Machine Learning, el humano es el que define cuáles son las variables - por ejemplo-)

DL trabaja como una caja negra, es decir, hay que esperar al final del procesamiento para verificar si clasificó bien o no.

Como en otros procesos, DL posee hiperparámetros que se pueden configurar.

En resumen, Deep Learning apunta -en su mayoría- a modelos que no son lineales o no paramétricos. A diferencia con Machine Learning, la fase de extracción la hace automáticamente la red y la cantidad de capas y neuronas por capas van a hacer la diferencia.

DL está basado en el “perceptrón”, que es la estructura de neurona más básica (diseñada en los años 50’)

Actualmente, existe Hardware que permite procesar la cantidad de capas, en tiempo prudente. Se puede utilizar GPU (unidad de procesamiento gráfica), TPU (unidad de procesamiento de Tensorflow de Google)

Conceptos sobre Deep Learning

Se basa en una idea simple: dados unos parámetros, hay una forma de combinarlos para predecir un cierto resultado.

Por ejemplo: sabiendo los píxeles de una imagen, habrá forma de saber qué número hay escrito.

Los datos de entrada van pasando secuencialmente por distintas “capas” en las que se aplican una serie de reglas de aprendizaje moduladas por una función peso.

Tras pasar por la última capa, los resultados se comparan con el resultado “correcto”, y se van ajustando los parámetros (dados por las funciones “peso”)

Esas redes son construcciones lógicas resultado de una serie de preguntas binarias (verdadero o falso) de las que se extrae un valor numérico; cada vez que incorpora un nuevo dato lo transfiere a esa red neuronal, y lo clasifica según la respuesta a dichas preguntas.

Limitaciones de Deep Learning

DL requiere gran volumen de datos de entrenamiento.

DL requiere tecnología de punta para procesamiento, tal es el caso de migrar procesamiento de CPU a GPU (o TPU)

Redes Neuronales

A continuación, algunas definiciones para las Redes Neuronales Artificiales, a partir de la comparación con las neuronas biológicas:

- Es un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.
- Un sistema de computación hecho por un gran número de elementos simples, elementos de proceso muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.
- Son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico” (J.R.Hilera, V.J. Martínez, 1995)

Modelo biológico de una Red Neuronal

La teoría y modelado de redes neuronales artificiales están inspirados en la estructura y funcionamiento de los sistemas nerviosos, donde la neurona es el elemento fundamental.

Una neurona consta de un cuerpo celular más o menos esférico, de 5 a 10 micrones de diámetro, del que salen una rama principal, conocida como axón, y varias ramas más cortas, llamadas dendritas. El axón puede producir ramas en torno a su punto de arranque y con frecuencia se ramifica extensamente cerca de su extremo.

NEURON

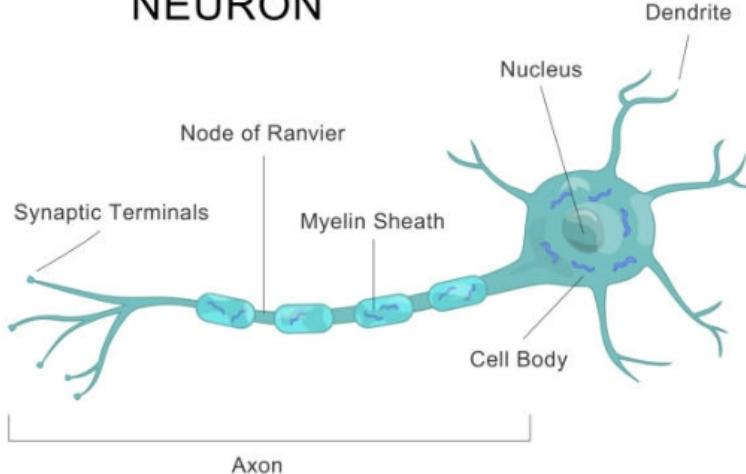


Ilustración 17. Estructura de una neurona biológica

Las interconexiones se realizan por medio de las ramas de salida (axones) que producen un número variable de conexiones (sinapsis) con otras neuronas. Las redes neuronales son sistemas de elementos de proceso simple pero muy interconectados.

Una neurona realiza un proceso, al generar variaciones de potencial eléctrico entre sus entradas y sus salidas. Por lo que al potencial eléctrico de la neurona se le puede considerar información y proceso al mecanismo que permite la modificación no lineal de esa información.



Ilustración 18. Cómo la neurona procesa la información y qué salida se obtiene.

Una neurona, puede -potencialmente- conectarse con miles de entradas y miles de conexiones de salida. Tomando en cuenta el enorme número de neuronas que posee un cerebro humano, se determina una cantidad de conexiones totales increíblemente grande, esto es del orden de (10^{15}) (F. James, 1997)

Desde un punto de vista computacional, cada neurona realiza un proceso “en teoría” simple, sin embargo el conjunto de estas tareas junto a la enorme telaraña de sus conexiones, genera una capacidad de cálculo capaz de resolver los problemas simbólicos más complejos que se presenten.

Las señales que se utilizan durante el proceso que realiza la neurona, son de naturaleza eléctrica y química. La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras que la señal que se transmite entre las terminales axónicas de una neurona y las dendritas de la neurona siguiente es un impulso químico. Este proceso se realiza mediante moléculas de sustancias transmisoras (neurotransmisoras) que fluyen a través de unos contactos especiales, llamados sinapsis, que tienen la función de receptor y que están localizados entre las terminales axónicas y las dendritas de la siguiente neurona. El espacio “sináptico” se mide entre 50 y 200 Ángstrom ($1\text{A} = 10^{-10}$ metros)

La generación de las señales eléctricas está íntimamente relacionada con la composición de la membrana celular. Estas señales se pueden simplificar del siguiente modo: la neurona, es capaz de mantener en su interior un líquido cuya composición difiere marcadamente de la composición del líquido exterior. La diferencia más notable se da en relación con la concentración de los iones sodio y potasio. El medio externo es unas 10 veces más rico en sodio que el interno, mientras que el medio interno es 10 veces más rico en potasio que el externo. Esta diferencia de concentración en iones sodio y potasio a cada lado de la membrana produce una diferencia de potencial de aproximadamente 70mV, negativa en el interior de la célula. Es lo que se llama potencial de reposo de la célula nerviosa.

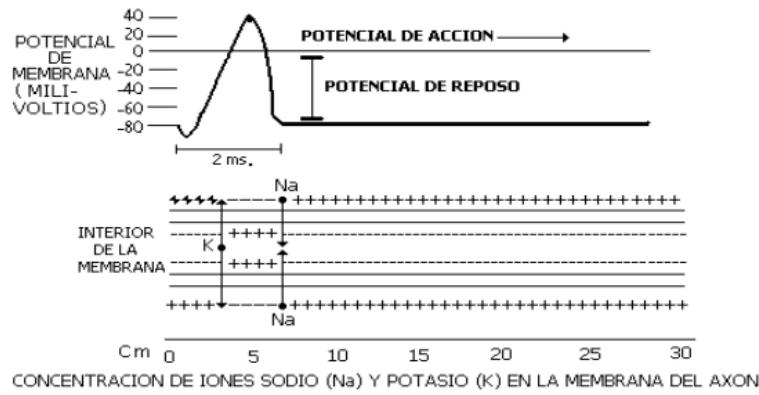


Ilustración 19. Señal de propagación del impulso eléctrico a lo largo del axón.

La llegada de señales procedentes de otra neurona a través de las dendritas (recepción de neurotransmisores) actúa acumulativamente, bajando ligeramente el valor del potencial de reposo. Dicho potencial modifica la permeabilidad de la membrana, de manera que cuando llega a cierto valor crítico comienza una entrada masiva de iones sodio que invierten la polaridad de la membrana.

La inversión del voltaje de la cara interior de la membrana cierra el paso a los iones sodio y abre el paso a los iones potasio hasta que se restablece el equilibrio en reposo. La inversión del voltaje, conocida como potencial de acción, se propaga a lo largo del axón y, a su vez, provoca la emisión de los neurotransmisores en las terminales axónicas.

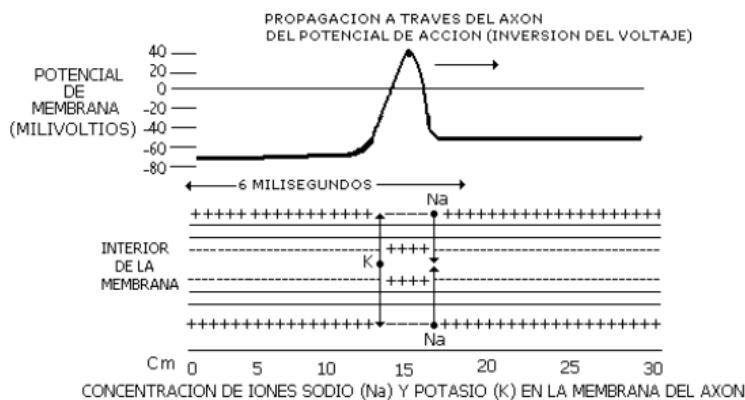


Ilustración 20. La inversión del voltaje, se propaga por el axón.

Cuando sucede la inversión de voltaje cierra el canal de sodio y abre el canal de potasio; la salida de iones potasio restablece rápidamente el potencial negativo.

Existen dos tipos de sinapsis:

- Sinapsis excitadoras: cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula postsináptica, facilitando la generación de impulsos a mayor velocidad.
- Sinapsis inhibidoras: cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos.

Una similitud directa entre la actividad sináptica y la analogía con las redes neuronales artificiales, es que las señales que llegan a la sinapsis son las entradas a la neurona, las cuales son ponderadas (atenuadas o amplificadas) a través de un parámetro, denominado peso, asociado a la sinapsis correspondiente. Estas señales de entrada pueden excitar a la neurona (sinapsis con peso positivo) o inhibirla (peso negativo). El efecto es la suma de las entradas ponderadas, si la suma es igual o mayor que el umbral de la neurona, entonces la neurona se activa (da una salida).

Esta situación es de todo o nada, es decir cada neurona se activa o no se activa; la habilidad de ajustar señales es un mecanismo de aprendizaje. Las funciones umbral integran la energía de las señales de entrada en el espacio y en el tiempo.

Red Neuronal Artificial

Modelo de una Red Neuronal Artificial

Las redes neuronales son modelos que intentan reproducir el comportamiento del cerebro, por lo tanto cualquier modelo de red neuronal consta de dispositivos elementales de proceso: neuronas.

Usualmente se pueden encontrar tres tipos de neuronas:

1. Aquellas que reciben estímulos externos, relacionadas con el aparato sensorial, que tomarán la información de entrada, conocidas como unidades de entrada.
2. Esta información se transmite a ciertos elementos internos que se ocupan de su proceso. Los cuales no tienen relación directa con la información de entrada ni con la de salida, estos elementos se denominan unidades ocultas.
3. Cuando finaliza el periodo de proceso, la información llega a las unidades de salida, cuya misión es dar la respuesta del sistema.

Cada i -ésima neurona está caracterizada en cualquier instante por un valor numérico conocido como valor o estado de activación $a_i(t)$ asociado a cada unidad; existe una función de salida f_i , que transforma el estado actual de activación en una señal de salida y_i . Esta señal se envía a través de las conexiones unidireccionales a otras unidades, en este lapso la señal se modifica de acuerdo con la sinapsis (el peso, w_{ji}) asociada a cada uno de ellos según una determinada regla. Por lo que se tienen tres casos:

1. $w_{ji} > 0$; Sinapsis excitadora.
2. $w_{ji} = 0$; No existe conexión.
3. $w_{ji} < 0$; Sinapsis inhibidora.

Las señales moduladas que han llegado a la unidad j -ésima se combinan entre sí generando la entrada total, Net_j , definida por la siguiente ecuación:

$$Net_j = \sum_i w_{ji} y_i(t)$$

y: Señal de salida

w: Peso

Net: Entrada total

Una función de activación, F , determina el nuevo estado de activación $a_j(t+1)$ de la neurona, teniendo en cuenta la entrada total calculada y el anterior estado de activación $a_j(t)$.

La actualización de los estados de las unidades puede ser de dos tipos:

- Modo asíncrono: las neuronas evalúan su estado continuamente, según les llega la información y lo hacen de forma independiente.
- Modo síncrono: la información llega de forma continua, pero los cambios se realizan simultáneamente, como si un reloj interno decidiera cuándo deben cambiar su estado.

Elementos básicos de una Red Neuronal Artificial

- **Unidades de proceso: Neurona artificial.** Su trabajo es simple y único, consiste en recibir las entradas de las células vecinas y calcular un valor de salida, el cual envía a todas las células restantes. El conjunto de neuronas cuyas entradas provienen de la misma fuente y cuyas salidas se dirigen al mismo destino, se denomina como capa o nivel. Si se tienen N -neuronas, se pueden ordenar de manera arbitraria y designar la j -ésima neurona como U_j .
- **Estado de activación.** En un modelo de red neuronal es necesario representar los estados del sistema en un tiempo t . Estos están representados por un vector de N -números reales $A(t)$, que representa el estado de activación del conjunto de neuronas de procesamiento. Cada elemento del vector representa la activación de una neurona en el tiempo t ; la activación de una neurona U_i en el tiempo t se designa por $a_i(t)$, definida por la ecuación:

$$A(t) = (a_1(t), a_2(t), \dots, a_i(t), \dots, a_N(t))$$

Todas aquellas neuronas que componen la red, se encuentran siempre en cierto estado. Los estados pueden ser: reposo y excitado (conocidos como estados de activación). Cada estado de activación tiene un valor, que puede ser discreto o continuo. Además, pueden ser limitados o ilimitados discretos. Cuando son discretos toman un conjunto pequeño de valores o bien valores binarios como 1 y 0, los cuales indican activo o pasivo respectivamente. El estado activo se caracteriza por la emisión de un impulso por parte de la neurona (potencial de acción). Y el estado pasivo significa que la neurona está en reposo. En otros modelos se considera un conjunto continuo de estados de activación que se encuentran en el intervalo [-1,1], generalmente siguiendo una función **sigmoidal**.

- **Función de salida o de transferencia.** A cada neurona U_i se encuentra asociada una función de salida $f_i(a_i(t))$, que transforma el estado actual de activación $a_i(t)$ en una señal de salida $y_i(t)$, lo que define la siguiente ecuación:

$$y_i(t) = f_i(a_i(t))$$

Entonces el vector que contiene las salidas de todas las neuronas en un instante t es: $Y(t) = (f_1(a_1(t)), f_2(a_2(t)), \dots, f_i(a_i(t)), \dots, f_N(a_N(t)))$

Existen cuatro funciones de salida típicas que determinan distintos tipos de neuronas:

- Función escalón.
- Función lineal.
- Función sigmoidal.
- Función Gaussiana.

FUNCIÓN ESCALÓN: La manera más fácil para definir la activación de una neurona es considerándola binaria. La función de transferencia escalón se asocia a neuronas binarias en las cuales, cuando la suma de las entradas es mayor o igual que el umbral de la neurona, la activación es 1; si es menor, la activación es 0 (o -1)

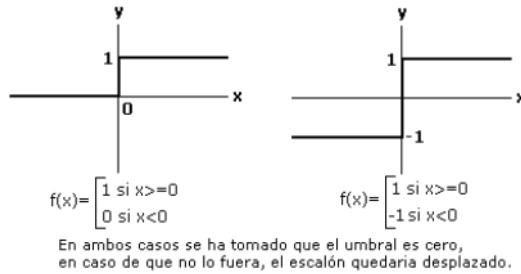


Ilustración 21. Función escalón

Función f escalón, cuando el conjunto de estados $E=[0,1]$

$$y_j(t+1) = \begin{cases} 1 & \text{si } [Net_j > \theta_j] \\ y(t) & \text{si } Net_j = \theta_{ji} \\ 0 & \text{si } [Net_j < \theta_j] \end{cases}$$

Función f escalón, cuando el conjunto de estados $E=[-1,1]$

$$y_j(t+1) = \begin{cases} +1 & \text{si } [Net_j > \theta_j] \\ y(t) & \text{si } Net_j = \theta_j \\ -1 & \text{si } [Net_j < \theta_j] \end{cases}$$

FUNCIÓN LINEAL: La función lineal o identidad responde a la expresión $f(x)=x$. En las neuronas con función mixta, si la suma de las señales de entrada es menor que un límite inferior, la activación se define como 0 (o -1); si la suma es mayor o igual que el límite superior, entonces la activación es 1. Cuando la suma de

entrada está comprendida entre ambos límites, superior e inferior, la activación se define como una función lineal de la suma de las señales de entrada.

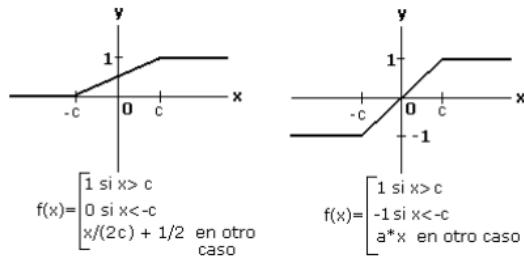


Ilustración 22. Función Lineal

Función f lineal: El conjunto de estados E contiene cualquier número real.

$$y_j(t+1) = Net_j - \theta_j$$

FUNCIÓN SIGMOIDAL: En una función sigmoidal el valor dado es cercano a uno de los valores asintóticos. Por lo tanto, el valor de salida está comprendido en la zona alta o baja del sigmoide. Cuando la pendiente es elevada, esta función tiende a una función escalón, sin embargo la importancia de la función sigmoidal es que su derivada es siempre positiva y cercana a cero para los valores grandes positivos o negativos, además toma su valor máximo cuando $x=0$. Esto hace que se puedan utilizar las reglas de aprendizaje definidas para las funciones escalón, con la ventaja de que su derivada está definida en todo el intervalo.

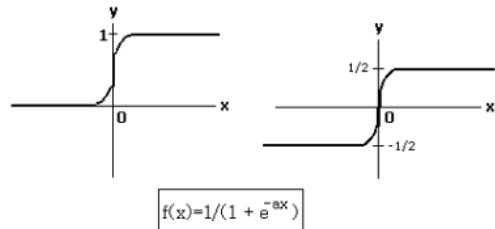


Ilustración 23. Función Sigmoidal

Función f sigmoidal: El conjunto de estados E está en un intervalo del eje real.

$$y_j(t+1) = \frac{1}{(1 + e^{-(Net_j - \theta_j)})}$$

FUNCIÓN GAUSSIANA: La función de transferencia Gaussiana indica que los centros y anchura de estas funciones pueden ser adaptados, lo que las hace más adaptativas que las funciones sigmoidales.

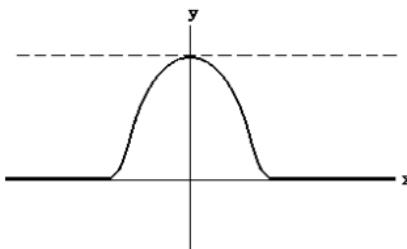


Ilustración 24. Función Gaussiana.

- **Conexiones entre neuronas.** Las conexiones que unen a las neuronas y que forman una Red Neuronal Artificial (RNA) tienen asociado un peso, que es el que hace que la red adquiera conocimiento. Usualmente se considera que el efecto de cada señal es aditivo, por lo tanto la entrada total que recibe una neurona, definido por Net_i es la suma del producto de cada señal individual por el valor de la sinapsis que conectan ambas neuronas:

$$Net_j = \sum_i^N w_{ji} * y_i(t)$$

Regla de propagación.supervised model

En esta ecuación se muestra el procedimiento a seguir para combinar los valores de entrada a una unidad, con los pesos de las conexiones que llegan a esa unidad. Es conocida como **regla de propagación**.

Algunas veces se utiliza una matriz W con los pesos w_{ji} que indica la influencia que tiene la neurona i sobre la neurona j .

W es un conjunto de elementos positivos, negativos o nulos; si w_{ji} es positivo, la interacción entre las neuronas i y j es **excitadora**, es decir siempre que la neurona i esté activada, la neurona j recibirá una señal de i que la active.

Si w_{ji} es negativo, la sinapsis será **inhibidora**, por lo que si i está activada, enviará una señal a j que la desactivará.

Si $w_{ji}=0$, indica que **no existe conexión** entre ambas neuronas.

- **Función o regla de activación.** Esta función F produce un nuevo estado de activación en una neurona a partir del estado a_i que existía y la combinación de las entradas con los pesos de las conexiones Net_i .

Para calcular el estado de activación siguiente $a_i(t+1)$ indicado en la siguiente ecuación, se aplica una función F llamada función de activación:

$$a_i(t+1) = F(a_i(t), Net_i)$$

Función de activación

Usualmente F es la función identidad, por lo que el estado de activación siguiente de una neurona es Net de la misma neurona en t . Por lo que la salida de una neurona j queda definido por la siguiente ecuación:

$$y_j(t+1) = f(Net_j) = f\left(\sum_{i=1}^N w_{ij}y_i(t)\right)$$

Estado de activación de la siguiente neurona

Normalmente la función de activación no está centrada en el origen del eje que representa el valor de la entrada neta, sino que existe un cierto desplazamiento debido a las características internas de la propia neurona y que no es igual en todas ellas. Este valor denotado como θ_j representa el umbral de activación de la neurona j . Entonces la función de activación queda mostrada en la siguiente ecuación:

$$y_j(t+1) = f(Net_j - \theta_j) = f\left(\sum_{i=1}^N w_{ij}y_i(t) - \theta_j\right)$$

Función de activación contemplando desplazamientos θ_j

Elementos básicos de una Red Neuronal Artificial

El aprendizaje es la modificación del comportamiento inducido por la interacción con el entorno y como resultado de experiencias conducente al establecimiento de nuevos modelos de respuesta a estímulos externos.

Biológicamente, la información memorizada en el cerebro está más relacionada con los valores sinápticos de las conexiones entre las neuronas, es decir, que el conocimiento se encuentra en las sinapsis; en el caso de las redes neuronales artificiales se considera que el conocimiento se representa con los pesos de las conexiones entre las neuronas. En realidad podríamos decir que se aprende modificando los valores de los pesos de la red.

Clasificación de algoritmos según entrenamiento

Algoritmos supervisados

Cada instancia (ejemplo) que recibe el algoritmo tiene indicado una clase o valor asociado. Por ejemplo, una imagen tiene asociada una categoría o un conjunto de categorías.

Se genera un modelo predictivo, basado en datos de entrada y salida.

Se tiene un conjunto de datos previamente etiquetado y clasificado; ya se sabe a qué grupo, valor o categoría pertenecen los ejemplos.

Con este grupo de datos, el cual llamamos datos de entrenamiento, se realiza el ajuste al modelo inicial planteado.

Así, el algoritmo va aprendiendo a clasificar las muestras de entrada, comparando el resultado del modelo, y la etiqueta real de la muestra, realizando las compensaciones respectivas al modelo de acuerdo a cada error en la estimación del resultado.



Ilustración 25. Aprendizaje supervisado

Ejemplos con distintos algoritmos

- Clasificación (variable discreta o cualitativa): Árboles de Decisión (Teorema de Bayes), Clasificador Bayesiano

- Regresión (variable continua, cuantitativa o numérica): Lineal Simple o Multivariada – con variables independientes continuas, o Politómicas: que admite varias categorías (encuestas de satisfacción: {mucho, poco, algo, casi}, {rangos etarios} , entre otros ejemplos) o Dicotómicas: admite 2 respuestas, 1 contraria a la otra.

Evaluación de Algoritmos supervisados

Se estima la exactitud del modelo basándose en un conjunto de datos de prueba.

Matriz de confusión: Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa las instancias en la clase real.

$M(C_i, C_i)$ son los casos correctamente clasificados.

$M(C_i, C_j)$ con $i \neq j$, son errores de clasificación. Idealmente, deberíamos tener valores en la diagonal principal y ceros en el resto de los lugares (donde: $i \neq j$)

La matriz de confusión permite medir la calidad del modelo.

Etiqueta de la clase	Predicciones C1	Predicciones C2	...	Predicciones Ck
Verdaderos C1	$M(C_1, C_1)$	$M(C_1, C_1)$...	$M(C_1, C_1)$
Verdaderos C2	$M(C_2, C_1)$	$M(C_2, C_2)$...	$M(C_2, C_k)$
...
Verdaderos Ck	$M(C_k, C_1)$	$M(C_k, C_2)$...	$M(C_k, C_k)$

Ilustración 26. Matriz de confusión.

Una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Uno de sus beneficios es que facilitan ver si el sistema está confundiendo dos clases (Jason Brownlee, 2016)

Errores Tipo I y Tipo II con ejemplos

El error de Tipo I (o error de tipo alfa (α), o falso positivo), es el error que se comete cuando el modelo rechaza la hipótesis nula (H_0 : el supuesto inicial) siendo ésta, verdadera en la población.

El error de Tipo II (o error de tipo beta (β), o falso negativo), es el error que se comete cuando el modelo no rechaza la hipótesis nula siendo ésta, falsa en la población.

Ejemplo con el diagnóstico de la enfermedad del paciente

El “Error tipo I” o “Falso positivo”, el algoritmo diagnosticó la enfermedad, pero el paciente no la tiene.

El “Error tipo II” o “Falso negativo”, el paciente tiene la enfermedad, pero el algoritmo no la diagnosticó.

Todas las métricas

		True condition	
		Total population	
Predicted condition	Total population	Condition positive	Condition negative
	Predicted condition positive	True positive (Power)	False positive (Type I error)
	Predicted condition negative	False negative (Type II error)	True negative

Tabla 3. Métricas. Predicciones vs. Realidad

True positive rate (TPR), Recall, Sensitivity, Probability of detection = Sum(True Positive)/Sum(Condition positive)	False positive rate (FPR), Fall-out, probability of false alarm = Sum(False positive) / Sum(Condition negative)
False negative rate (FNR), Miss rate = sum(False negative) / Sum(Condition positive)	Specificity (SPC), Selectivity, True negative rate (TNR) = Sum(True negative) / Sum(Condition negative)

Tabla 4: métricas más comunes.

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}$$

$$\text{Recall} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$$

$$\text{F1-score} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

Tabla con las métricas más comunes (ii)

Exactitud (Accuracy)

Es el porcentaje de muestras del conjunto de test que son correctamente clasificadas por el modelo (los que fueron bien clasificados) - Accuracy = la suma de todos los TRUE (de la diagonal principal) / el valor de las 4 celdas.

La Exactitud (en inglés, “Accuracy”) se refiere a lo cerca que está el resultado de una medición del valor verdadero. En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. Se representa como la proporción de resultados verdaderos (tanto verdaderos positivos (VP) como verdaderos negativos (VN)) dividido entre el número total de casos examinados (verdaderos positivos, falsos positivos, verdaderos negativos, falsos negativos)

Accuracy: This function is an indication of how close the measured value is to the true value, and is defined in Eq. (12).

$$Accuracy = \left(\frac{1}{N} \sum_{j=1}^N |P_j - A_j| \right) * 100 \quad (12)$$

[View Source](#) 

where $|P_j - A_j| \leq 1$.

Definición: Estimador Accuracy

Sensibilidad (Recall o TPR)

La sensibilidad y la especificidad son dos valores que nos indican la capacidad de nuestro estimador para discriminar los casos positivos, de los negativos. La sensibilidad se representa como la fracción de verdaderos positivos, mientras que la especificidad, es la fracción de verdaderos negativos.

La Sensibilidad (“Recall” o “Sensitivity”), también se conoce como Tasa de Verdaderos Positivos (True Positive Rate) ó TP. sirve para saber si no están perdiendo positivos. Es la proporción de casos positivos que fueron correctamente identificadas por el algoritmo. Es la cobertura: “de todos los positivos que tenía que encontrar, ¿cuántos encontré?”

Por el otro lado, la especificidad , también conocida como la Tasa de Verdaderos Negativos, (“true negative rate”) o TN. Se trata de los casos negativos que el algoritmo ha clasificado correctamente. Expresa cuan bien puede el modelo detectar esa clase.

Precisión (Precision)

Es la proporción de verdaderos positivos dividido entre todos los resultados positivos (tanto verdaderos positivos, como falsos positivos).

La Precisión (en inglés “Precision”) Se refiere a la dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud. Cuanto menor es

la dispersión mayor la precisión. Se representa por la proporción de verdaderos positivos dividido entre todos los resultados positivos (tanto verdaderos positivos, como falsos positivos).

La precisión ayuda a conocer qué tan seguro estamos de los verdaderos positivos. En forma práctica es el porcentaje de casos positivos detectados.

La precisión es un estadístico, útil cuando los “datasets” son simétricos (la cantidad de casos de la clase 1 y de las clase 2 tienen magnitudes similares)

F1-score

Esta es otra métrica muy empleada porque nos resume la precisión y sensibilidad en una sola métrica. Por ello es de gran utilidad cuando la distribución de las clases es desigual.

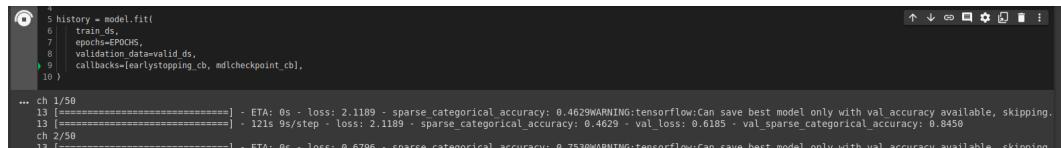
El indicador F1 de la matriz de confusión es útil si se tiene una distribución de clases desigual.

¿Qué métrica se ajusta mejor al modelo?

El rendimiento del modelo se puede mejorar después de analizar la precisión y la exactitud (S. Jha et al. 2019) Para identificar el algoritmo más eficiente sobre la base de diferentes valores de accuracy, el artículo de referencia presenta los resultados de evaluaciones de diferentes medidas de precisión de predicción y precisión (sin ninguna reducción de características), siendo “Accuracy” la métrica más representativa pero no la única que se debe analizar, apoyada en “Recall” y “Precision” (Keras metrics, 2022)

¿Cuántas EPOCHS es necesario ejecutar?

Por el lado de Tensorflow, cuando el modelo está en su etapa de entrenamiento, se puede guardar la mejor EPOCH (definiendo “mejor” a la que tenga el máximo valor de la medida elegida) – en este caso, se está resolviendo la métrica “Accuracy”



```
5 history = model.fit(
6     train_ds,
7     epochs=EPOCHS,
8     validation_data=valid_ds,
9     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
10 )
...
ch 1/50
13 [=====] - ETA: 0s - loss: 2.1189 - sparse_categorical_accuracy: 0.4629WARNING:tensorflow:Can save best model only with val accuracy available, skipping.
ch 2/50
13 [=====] - ETA: 0s - loss: 2.1189 - sparse_categorical_accuracy: 0.4629 - val_loss: 0.6185 - val_sparse_categorical_accuracy: 0.8458
...  
ch 1/50
13 [=====] - ETA: 0s - loss: 0.6706 - sparse_categorical_accuracy: 0.7530WARNING:tensorflow:Can save best model only with val accuracy available, skipping.
```

Ilustración 27: se puede guardar el mejor modelo si se calcula accuracy.

El hecho de no involucrar en el desarrollo a dicha métrica, dificulta obtener conclusiones referidas al rendimiento del modelo.

Relación entre Sensibilidad (Recall), Precision, y Errores Tipo I y II

Se incorporaron las métricas más representativas, para evaluar los siguientes cuatro casos posibles para cada clase:

Alta precisión y alto recall: el modelo escogido maneja perfectamente esa clase.

Alta precisión y bajo recall: el modelo escogido no detecta la clase muy bien, pero cuando lo hace es altamente confiable.

Baja precisión y alto recall: El modelo escogido detecta bien la clase, pero también incluye muestras de la otra clase.

Baja precisión y bajo recall: El modelo escogido no logra clasificar la clase correctamente.

A efectos prácticos, a igualdad de True Positive:

- 0,8 precision (más alta que recall) y 0,6 recall (no tan alta), hay error Tipo 2 (o Beta)

- 0,6 precision (no tan alta) y 0,8 recall (más alta que precisión), hay error Tipo 1 (o Alfa)
- Una F-measure alta, no es síntoma de un buen modelo. Hay que ver recall (cobertura) y precision.

Todos valores tienen que estar por encima de 0,7 para ser aceptable.

Si es mejor tener falsos positivos que falsos negativos, conviene usar una sensibilidad alta (Recall), cuando la aparición de falsos negativos le resulta inaceptable pero no le importa tener falsos positivos adicionales (falsas alarmas)

$\text{Accuracy (ACC)} = (\text{Sum(TP)} + \text{Sum(TN)}) / \text{Sum (Total population)}$	$F1 \text{ score} = 2 / ((1/\text{Recall}) + (1/\text{Precision}))$
$\text{Prevalence} = \text{Sum(Condition positive)} / \text{Sum(Total population)}$	$\text{Precision, Positive predictive value (PPV)} = \text{Sum(TP)} / \text{Sum(Predicted condition positive)}$
$\text{False discovery rate (FDR)} = \text{Sum(FP)} / \text{Sum(Predicted condition negative)}$	$\text{Negative predictive value (NPV)} = \text{Sum(TN)} / \text{Sum(Predicted condition negative)}$
$\text{Positive likelihood ratio (LR+)} = \text{TPR} / \text{FPR}$	$\text{Negative likelihood ratio (LR-)} = \text{FNR} / \text{TNR}$
$\text{Diagnostic odds ratio (DOR)} = \text{LR+} / \text{TNR}$	$\text{False omission rate (FOR)} = \text{Sum(FN)} / \text{Sum(predicted condition negative)}$

Tabla 5: Resto de métricas usadas en aprendizaje supervisado.

Métricas desde la librería Keras

De acuerdo con la nota de divulgación de Keras (Keras-2.0-release-notes, 2020) ya no están disponibles las métricas “Precision” y “Recall” -entre otras librerías- a través de la función Metrics de la librería Keras (directamente)

Losses & metrics

- The `objectives` module has been renamed `losses`.
- Several legacy metric functions have been removed, namely `matthews_correlation`, `precision`, `recall`, `fbeta_score`, `fmeasure`.

Prácticamente, las mismas existen, se pueden instanciar, pero en tiempo de ejecución van a fallar. En consecuencia, se utilizó `sklearn.metrics`, asegurando que se pueden calcular (Keras-metrics 1.1.0, 2020)

Algoritmos no supervisados

No requieren una clase para funcionar. Sólo se valen de las instancias con sus características. Hay que entrenar, para encontrar patrones que permita hallar similitudes por proximidad entre atributos.

Encuentran patrones y retornan los tipos encontrados.

Entonces, primero tenemos un modelo no supervisado cuando tenemos 1 dataset sin ningún contexto. Luego, el algoritmo detecta grupos, y forma clusters. Al final, se obtiene un modelo supervisado.

Ejemplos: Clustering, Reducción de la dimensionalidad, Análisis de Componentes Principales, Descomposición por valores singulares.

No se abordará el tema en este documento debido a que se trata de algoritmos destinados a resolver una problemática diferente.

CAPÍTULO 3: ARQUITECTURA

Arquitectura de Referencia

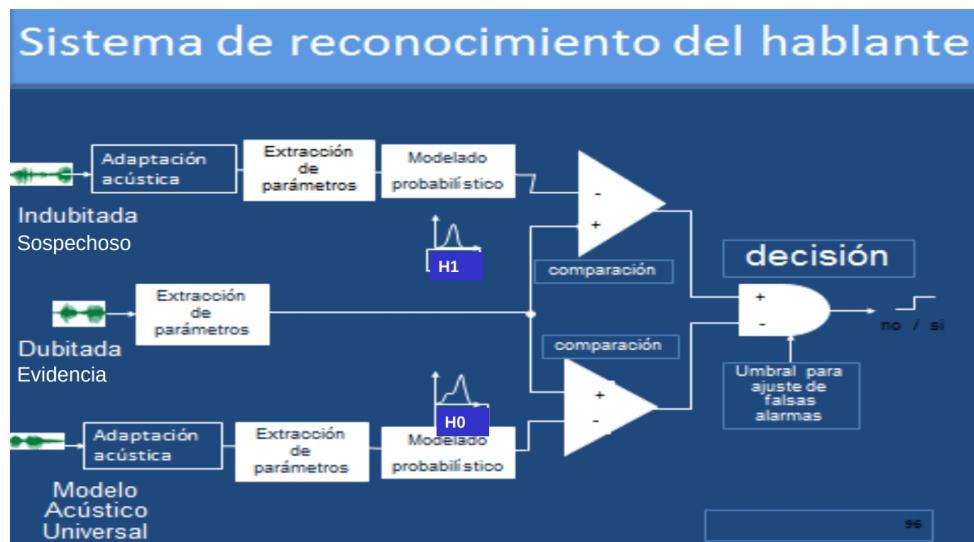


Ilustración 28. Sistema de reconocimiento del hablantes - CONICET , 2016.

Arquitectura de Sistema

<https://googlecloudcheatsheet.withgoogle.com/architecture?link=e87a3680-912e-11ed-abd6-89d652aadb6>

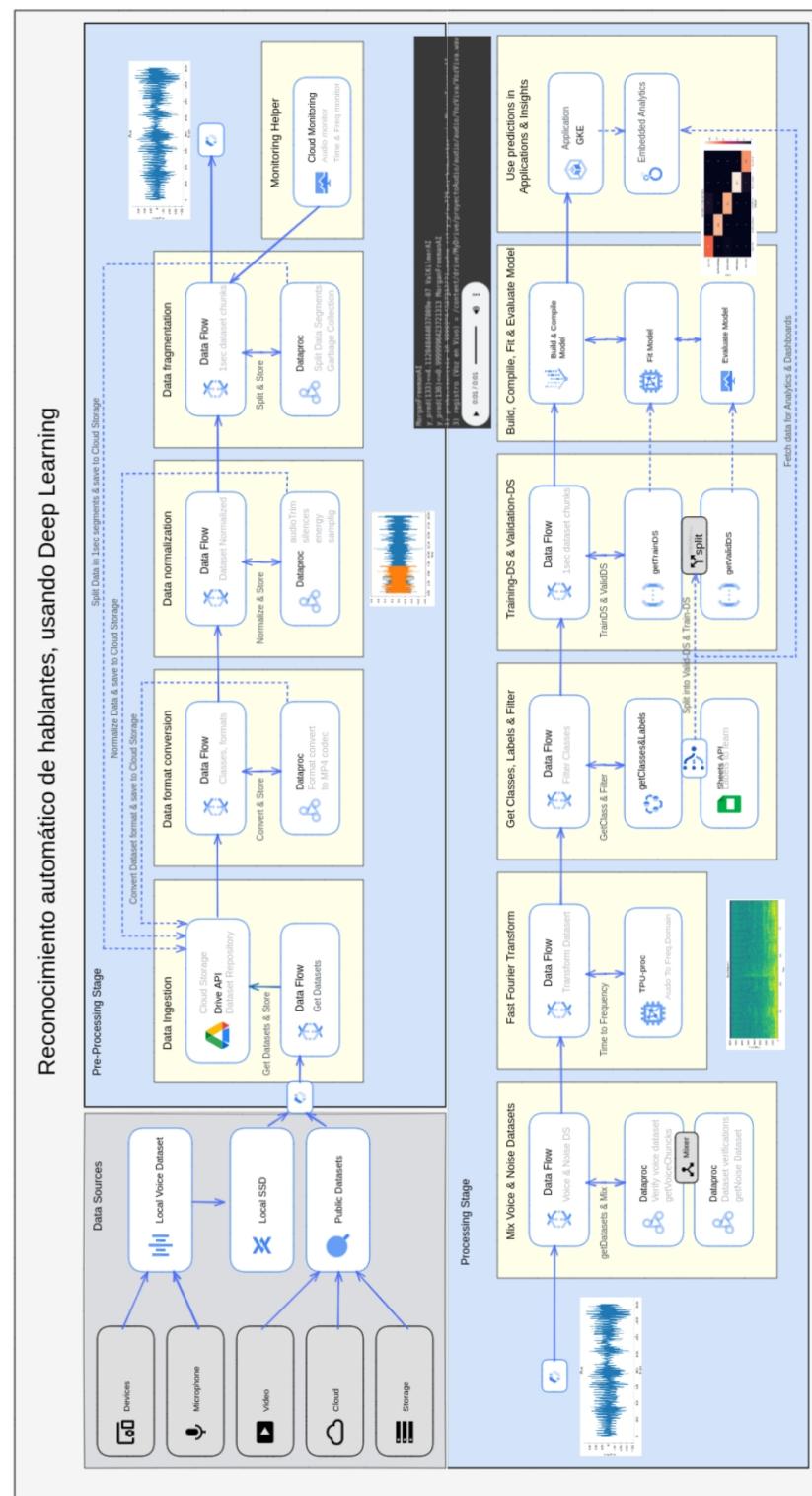


Ilustración 157. Arquitectura de Sistema, Reconocimiento de hablantes utilizando deep-learning

CAPÍTULO 4: MARCO METODOLÓGICO

Reconocimiento de Voz

En este apartado, se abordan técnicas de Reconocimiento de voz, como preámbulo al desarrollo de la Identificación de Hablantes.

El objetivo es estar familiarizado con el entorno de trabajo, la configuración del mismo, la instalación de librerías, la ejecución del código y la visualización de los resultados.

Estos pequeños proyectos relacionados, están disponibles para ser consultados a través de este enlace: [voiceRecognition001.ipynb](#)

O bien, desde: https://github.com/jmiguez/reconocimiento_de_hablantes

Para esto, se utilizó como entorno de desarrollo: Anaconda Navigator 2.0.3 con Jupyter Notebook 6.3.0 , y python 3.8.8.

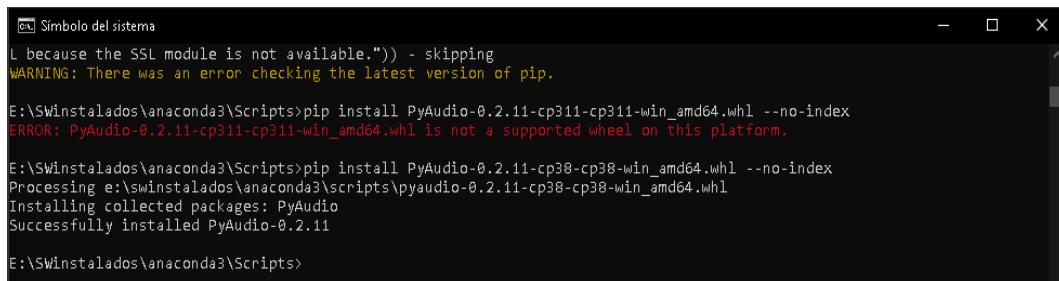
El motivo de usar este entorno, fue porque resultó más simple implementar las librerías con los periféricos en un ordenador local, en lugar de solucionar los problemas relacionados con la utilización de periféricos de audio en entornos en la nube (cloud, como Google Colab)

Primer proyecto

En este proyecto inicial, se reproduce una frase escrita, mediante una voz.

Como pre-requisito, hay que instalar pyaudio para la versión: \$python --version
⇒ 3.8.8, en el path: \$where python.

PyAudio es una rutina para poder usar el micrófono, permite configurar la cancelación de ruidos, y la salida de audio.



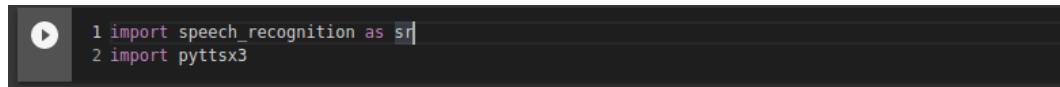
```
Symbolo del sistema
L because the SSL module is not available.")) - skipping
WARNING: There was an error checking the latest version of pip.

E:\SWinstalados\anaconda3\Scripts>pip install PyAudio-0.2.11-cp311-cp311-win_amd64.whl --no-index
ERROR: PyAudio-0.2.11-cp311-cp311-win_amd64.whl is not a supported wheel on this platform.

E:\SWinstalados\anaconda3\Scripts>pip install PyAudio-0.2.11-cp38-cp38-win_amd64.whl --no-index
Processing e:\swinstalados\anaconda3\scripts\pyaudio-0.2.11-cp38-cp38-win_amd64.whl
Installing collected packages: PyAudio
Successfully installed PyAudio-0.2.11

E:\SWinstalados\anaconda3\Scripts>
```

Al comienzo hay que importar las librerías:



```
1 import speech_recognition as sr
2 import pyttsx3
```

Speech_recognition

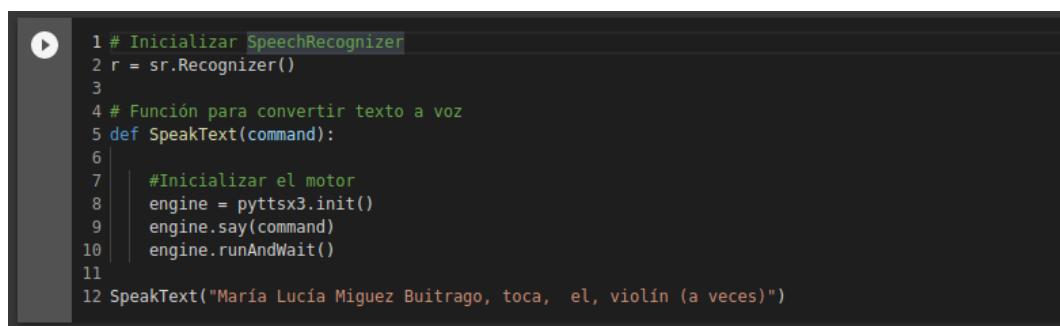
(<https://pypi.org/project/SpeechRecognition/>)

Librería para realizar reconocimiento de voz, con soporte para varios motores y APIs, en línea y fuera de línea.

Pyttsx3

(<https://pypi.org/project/pyttsx3/>)

Librería de texto a voz (TTS) para Python 2 y 3. Funciona sin conexión a Internet ni demora. Admite múltiples motores TTS, incluidos Sapi5, nsss y espeak.



```
1 # Inicializar SpeechRecognizer
2 r = sr.Recognizer()
3
4 # Función para convertir texto a voz
5 def SpeakText(command):
6
7     #Inicializar el motor
8     engine = pyttsx3.init()
9     engine.say(command)
10    engine.runAndWait()
11
12 SpeakText("María Lucía Miguez Buitrago, toca, el, violín (a veces)")
```

A continuación, la lógica principal:

En la línea 2, se instancia la librería speech_recognition.

En la línea 12, se invoca a una función, con un argumento que representa a un texto que será reproducido por una voz.

La función, definida en la línea 5, recibe como parámetro el argumento de la línea 12. Se inicializa, se le dá la orden de reproducir, y se ejecuta la instrucción.

El resultado es escuchado a través de los parlantes de la laptop usada para ejecutar los comandos escritos en Jupyter-Notebook. Se notó un acento extranjero (inglés) al momento de escuchar el mensaje.

Segundo proyecto

Registrar la voz de una persona a través de la entrada de audio (micrófono), mostrar en pantalla la frase registrada, y reutilizar el proyecto anterior para reproducir el mensaje mediante una voz.

Como primera medida, hay que comentar la línea 12, que es la llamada a la función que reproduce el audio.

```
▶ 1 # Inicializar SpeechRecognizer
 2 r = sr.Recognizer()
 3
 4 # Función para convertir texto a voz
 5 def SpeakText(command):
 6
 7     #Inicializar el motor
 8     engine = pyttsx3.init()
 9     engine.say(command)
10     engine.runAndWait()
11
12 # SpeakText("María Lucía Miguez Buitrago, toca, el, violín (a veces)")
```

En un nuevo bloque, en la línea 4, se definió la fuente de entrada de audio.

Con la línea 7, aseguramos que el usuario pueda leer una advertencia mientras se inicializa el entorno de trabajo. El mismo permite ejecutar los ajustes ambientales

de la línea 8 para normalizar el ruido de fondo (durante 2 segundos) y así lograr la interpretación de la voz de manera más efectiva.

Al ejecutar la línea 12, la notebook queda a la espera de que el usuario registre la voz a través del micrófono, y mientras no se detecte silencio, la entrada de audio quedará activa. El audio capturado por la entrada, es transformado a texto en la línea 15, y editado en minúsculas en la línea 16.

```
▶ 1 # Usar el micrófono como fuente de entrada
 2 #pip install pyaudio
 3
 4 with sr.Microphone() as source2:
 5     # esperar por unos instantes para que SpeechRecognizer ajuste
 6     # umbrales de energía basados en el nivel envolvente de ruido
 7     print("Silencio por favor, calibrando ruidos de fondo")
 8     r.adjust_for_ambient_noise(source2, duration=2)
 9     print("Calibrado, ahora puede hablar.....")
10
11     # escuchar la entrada del usuario
12     audio2 = r.listen(source2)
13
14     #Usar Google para reconocer audio
15     MyText = r.recognize_google(audio2)
16     MyText = MyText.lower()
17
18     print("Usted dijo: " + MyText)
19     SpeakText(MyText)
```

En la línea 18, el texto guardado en minúsculas se muestra por consola.

Finalmente, la variable que contiene el texto, es usada como argumento para llamar a la función que reproduce el texto de una variable del ejercicio anterior.

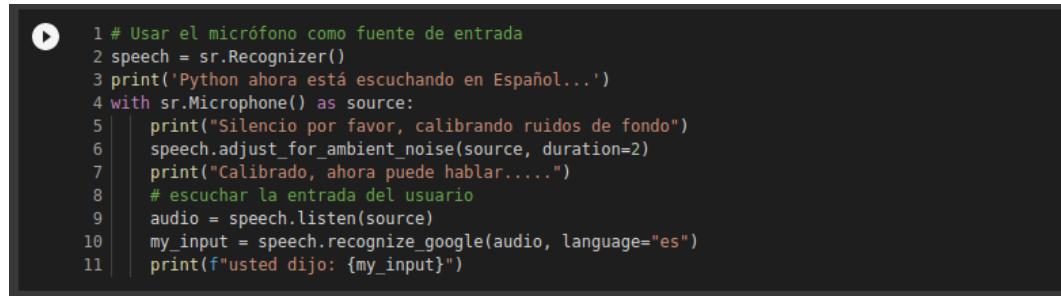
El intérprete que transforma la señal de audio a texto, está entrenado en lenguaje inglés. Motivo por el cual, el texto en castellano muchas veces no es compatible con la frase que le usuario emitió.

Tercer proyecto

Igual al problema anterior, pero haciendo el procesamiento en otro idioma. En este caso, “español”.

A diferencia del caso anterior, en este nuevo bloque se incorporó el reconocimiento de una lengua diferente a la lengua por defecto (en este caso, “es” = español)

A partir de este cambio, el texto capturado e impreso por consola en la línea 11, coincide con la frase emitida por el usuario en español.



```
1 # Usar el micrófono como fuente de entrada
2 speech = sr.Recognizer()
3 print('Python ahora está escuchando en Español...')
4 with sr.Microphone() as source:
5     print("Silencio por favor, calibrando ruidos de fondo")
6     speech.adjust_for_ambient_noise(source, duration=2)
7     print("Calibrado, ahora puede hablar.....")
8     # escuchar la entrada del usuario
9     audio = speech.listen(source)
10    my_input = speech.recognize_google(audio, language="es")
11    print(f"usted dijo: {my_input}")
```

Esto aplica a la transformación de la captura de audio a texto. No así a la reproducción, que hasta este momento, sigue siendo en lengua inglesa.

Cuarto proyecto

Convertir un registro de voz en idioma español, a texto, y reproducir el audio del texto.

En este caso, se utiliza el control de errores para manejar las excepciones que el método que convierte de voz a texto pueda informar.

En este bloque, la línea 22 propaga un mensaje al usuario en caso de que la conversión de audio a texto no sea exitosa.

Entre la línea 26 y la línea 32, el texto fue convertido a un archivo de audio en el idioma seleccionado, y en la línea 33 el archivo es reproducido de la librería de audio importada al comienzo de estos proyectos.

```

1 # Convertir texto a voz en el idioma que quiero
2 from io import BytesIO
3 import speech_recognition as sr
4 from gtts import gTTS
5 from pydub import AudioSegment
6 from pydub.playback import play
7
8 # Usar el micrófono como fuente de entrada
9 speech = sr.Recognizer()
10 print('Python ahora está escuchando en Español...')
11 with sr.Microphone() as source:
12     print("Silencio por favor, calibrando ruidos de fondo")
13     speech.adjust_for_ambient_noise(source, duration=1)
14     while True:
15         try:
16             print("Calibrado, ahora puede hablar.....")
17             # escuchar la entrada del usuario
18             audio = speech.listen(source)
19             my_input = speech.recognize_google(audio, language="es")
20             break
21         except sr.UnknownValueError:
22             print("No entendí lo que dijo !")
23 print(f"usted dijo: {my_input}")
24
25
26 tts = gTTS(text=my_input, lang="es")
27 # Crear un archivo temporal
28 voice = BytesIO()
29 # Guardar la salida en un archivo de sonido
30 tts.write_to_fp(voice)
31 # Reproducir el archivo de voz
32 voice.seek(0)
33 play(AudioSegment.from_mp3(voice))

```

Hasta aquí, se elaboró material de prueba para presentar un entorno de trabajo local, su configuración, ejecución de líneas de código relacionadas con librerías de audio, y obtención de resultados tangibles.

Análisis del Método Automático

En esta sección se desarrolla el modelo para clasificar hablantes, basado en el dominio de frecuencias de registros de voz, obtenidas a partir de Transformadas Rápidas de Fourier (FFT).

El objetivo es crear una Red neuronal convolucional de 1 dimensión (CNN-1D), con conexiones residuales para la clasificación de audio, resultando en la detección automática de hablantes.

El método automático consta de las siguientes etapas:

- Etapa de pre-procesamiento. Para proveer un set de datos normalizado, adaptado para el consumo de la siguiente etapa. Técnicamente, esta etapa no requiere procesadores GPU ni TPU, pero a partir del próximo paso, es recomendable usar GPU o TPU (no es bloqueante ya que el poder de procesamiento no altera el resultado, pero prolonga el tiempo de respuesta)
- Etiquetar el conjunto de datos de muestras de voz de los diferentes hablantes, con el hablante como etiqueta. Se agrega ruido de fondo en cada muestra para aumentar la cantidad de datos.
- Obtener la FFT de todas las muestras.
- Entrenar la red convolucional 1D
- Predecir al hablante correcto.

Nota: Las muestras de ruido en el conjunto de datos se deben volver a muestrear a una frecuencia de 16000Hz. Para hacer esto, se debe tener instalado ffmpeg (que es una colección de software libre que permite grabar, convertir y hacer streaming de audio y vídeo)

Librerías utilizadas

pytube

Es una librería ligera, abierta y gratuita desarrollada en Python (PyTube3, 2022) que permite descargar videos de YouTube. Esta librería incluye la utilidad de línea de comando, permitiendo descargar videos desde la terminal de comandos, además, no

tiene dependencias de terceros. A continuación se presenta algunas de sus características:

- Soporta streams progresivos y tipo DASH.
- Puede descargar una playlist completa.
- Fácil registro de callbacks como `on_download_progress` y `on_download_complete`.
- Interfaz de línea de comando incluida.
- Soporta caption track.
- Salida de caption tracks en formato .srt.
- Capacidad de captura de thumbnail URL.
- Código fuente extensamente documentado.
- Sin dependencias de terceros.

sox y pysox

SoX (*SoX - Sound eXchange*, 2015) es una utilidad de línea de comando multiplataforma (Windows, Linux, MacOS, etc.) usada para procesamiento de audio, puede convertir varios formatos de archivos de audio en otros formatos, convertir tasas de muestreo. También puede aplicar varios efectos de sonido a estos archivos, adicionalmente puede reproducir y grabar archivos de audio en muchas de las plataformas.

Pysox (*Pysox Documentation*, 2021) es la librería construida en Python alrededor de las funcionalidades propias de SoX (*Source code for sox.transform*, 2016)

ffmpeg

Ffmpeg (*sox and ffmpeg.ipynb*, 2022) es un convertidor muy rápido de audio y video, incluso desde una fuente en vivo de audio/video. Puede convertir entre archivos con tasas de muestreo arbitrarias y cambiar el tamaño del video con una alta calidad. Puede ser usado para convertir WAV a MP3 y viceversa.

Pydub

Pydub (*Pydub*, 2022) es una librería de Python que trabaja solo con archivos .wav. Al hacer uso de la librería se puede reproducir, dividir, integrar o editar los archivos de audio *.wav.

Las siguientes son las funcionalidades que se pueden ejecutar en la pydub:

- Reproducir archivos de audio.
- Obtener información del archivo como longitud de canales.
- Incrementar/disminuir el volumen de un archivo .wav determinado.
- Integrar dos o más archivos de audio.
- Exportar un archivo de audio.
- Dividir un archivo de audio.

Xlrd

La librería xlrd es usada para extraer datos de una hoja de cálculo, solo trabaja con archivos de extensión .xls. Es usado para leer, escribir o modificar datos, permite

navegar entre varias páginas de la hoja de cálculo y extraer datos dependiendo de ciertos criterios definidos.

Ipython.display

Es una API pública usada para desplegar herramientas en Ipython (*Using Audio in Ipython*, 2022), dentro de sus clases se encuentra una para audio, con la cual se crea un objeto que ante una entrada o una función retornará como resultado controles de audio que se mostrarán en el frontend (*pyAudioAnalysis*, 2015)

Librosa

Es un paquete de Python para el análisis de música y audio. Provee los bloques funcionales que permiten crear sistemas de recuperación de información musical. Dentro de sus capacidades está la de cargar audio desde el disco, computar varias representaciones de espectrogramas y una variedad de herramientas usadas comúnmente para el análisis musical.

Pandas

Es una librería de Python especializada en el manejo y análisis de estructuras de datos. Sus principales características son:

- Definir nuevas estructuras de datos basadas en arreglos de la librería NumPy pero con nuevas funcionalidades.
- Permite leer y escribir de varias fuentes de datos como archivos csv, Excel, Bases de datos SQL.
- Acceso a datos mediante índices o nombres para filas y columnas.
- Métodos para reordenar, dividir y combinar conjuntos de datos.

- Trabaja con datos de series de tiempos.

Keras

Es una API diseñada para la implementación y manejo de redes neuronales de forma sencilla, desarrollada en Python y es de código abierto.

Keras maneja redes neuronales estándar, convolucionales y redes neuronales recurrentes. Entre sus casos de uso, se aprecian métodos para procesamiento de archivos de voz, y procesamiento de lenguaje natural.

Keras provee ejemplos escritos para Jupyter notebook, que pueden ser ejecutados fácilmente en Google Colab, que es un ambiente que no requiere configuración inicial y corre en la nube. Google Colab incluye procesadores GPU y TPU, otorgando mayor rendimiento (respecto a un procesador CPU)

Matplotlib y Numpy

Matplotlib permite graficar los datos en figuras, las cuales puede tener uno o muchos ejes y un área donde los puntos pueden ser especificados en términos de coordenadas x-y-..

Dentro de las entradas esperadas para la ejecución de la función de ploteo se encuentran arreglos provenientes de la librería de NumPy (*Plot A Wave Audio File*, 2020)

La combinación de estas dos librerías (*Basic Sound Processing in Python*, 2015) proveerá la capacidad de plotear un archivo de audio de extensión .wav en Python, pero se debe asegurar que el archivo sea monocanal.

essentia.standard.Monoloader

Esta es una librería de código abierto para obtener información de archivos de música para análisis de audio. dentro de las funcionalidades con las que cuenta essentia (*Essentia Python tutorial*, 2022) están:

- Cargar archivos de audio.
- Ejecutar operaciones numéricas como FFT.
- Graficar resultados.
- Exportar los resultados del análisis a un archivo.

Particularmente, Monoloader permite obtener un nuevo archivo de audio mezclado y con un re-muestreo a una tasa de muestreo específica.

Equipo de procesamiento Local

A efectos de realizar el procesamiento en un terminal local, se utilizó el siguiente entorno:

Hardware

- Procesador: AMD® A4-4000 apu with radeon(tm) hd graphics × 2
- Capacidad de disco: 980,2 GB
- Memoria: 10 GiB
- Nombre y Tipo de SO: Ubuntu 22.04 LTS 64 bits
- Gráficos: AMD® Aruba

Software

- python3

- pip + pip3
- Anaconda Navigator 2.2.0 con Jupyter Notebook

A efectos de realizar el procesamiento en Anaconda / Jupiter-notebook - El desarrollo persiste en el repositorio público, cuya URL de acceso es:
https://github.com/jmiguez/reconocimiento_de_hablantes

Procesamiento en la nube

A efectos de realizar el procesamiento en la nube, se utilizó Google Colab - El desarrollo se puede descargar desde el repositorio antes mencionado:
[procesamiento_V0080101.ipynb](#)

Espacio de almacenamiento en Google Colab (por defecto, usando GPU) = 15GiB.

Conjunto de datos

Conjunto preliminar de archivos de voz de hablantes

Como puntapié inicial para este proyecto, es necesario contar con registros de audio de hablantes para ser procesados rápidamente y comenzar a validar si el camino a seguir es el correcto.

Keras es una librería que provee APIs para el procesamiento de voz. Su documentación oficial, provee un set de datos que sí se puede utilizar como punto de partida. El mismo corresponde a registros de voz de 5 figuras públicas (denominados en dicho portal como “líderes prominentes”), en lengua inglesa -ya se observó, que el procesamiento de voz podría depender del idioma como también de otros factores intrínsecos de la voz-

El set de datos contiene, para cada una de 5 personas, 1500 registros de audio en archivos de muy corta duración.

También incluye 7 archivos con diferentes fuentes de ruido, para mezclarlos con las pistas de voz: <https://www.kaggle.com/datasets/kongaevans/speaker-recognition-dataset?resource=download>

Las muestras de ruido de fondo, se distribuyeron en 2 carpetas con un total de 6 archivos. Estos archivos duran más de 1 segundo (originalmente no se están muestreados a 16000 Hz, por eso se volverán a muestrear a 16000 Hz).

Durante este proyecto, se usarán los 6 archivos de ruido, para crear muestras de ruido de 1 segundo de duración que se usarán para el entrenamiento.

Las 2 categorías se ordenan en 2 carpetas:

- Audio: con todas las carpetas de muestra de voz por hablante.
- Ruido: con todas las muestras de ruido.

Conjunto de archivos de audio de hablantes sin ruido

Una vez que el modelo fue validado, se procedió a obtener un set de datos adaptado al ambiente local. Por eso, como primera instancia, se dirigió el enfoque a los congresos TEDx. Éstos son organizados de manera independiente bajo una licencia exclusiva de TED que puede ser obtenida por quien acuerde de antemano seguir sus principios. Los conferenciantes deben renunciar a los derechos de autor sobre los materiales generados, para que TED pueda editarlos y distribuirlos bajo su licencia. Están disponibles públicamente en internet, por ejemplo, a través de un canal de YouTube.

Lo curioso de estos espacios, es la duración de las charlas, que duran como máximo 18 minutos. Según afirman, “son lo suficientemente largas como para ser serias”, pero también “lo suficientemente cortas para mantener la atención de la audiencia”. El orador, se encontrará hablando durante el tiempo que dure la presentación, sin interrupciones, y sin ruido de fondo. Es un ambiente ideal para obtener diferentes muestras de voz en crudo sin necesidad de identificar diferentes hablantes en un mismo archivo de audio. Es el tipo de registro de voz que se necesita para validar que el modelo esté funcionando correctamente.

Conjunto de archivos de voces en idioma español, para pre-procesar

Luego de haber trabajado con archivos sin ruido, la siguiente etapa es la obtención de registros de audio en crudo, en donde las voces pertenecen a personajes de público acceso, en lengua española, donde la duración de la exposición de cada hablante no es fija, y existen interrupciones, ruidos, los parámetros de audio no son homogéneos y varían entre cada hablante.

En este sentido, la opción es la captura de los canales de las cámaras de diputados y senadores de la Nación Argentina.

Al ser figuras públicas, es posible obtener registros de audio de alguno de estos hablantes: tanto en la actualidad, como también registros de audio de hace algunos años atrás.

El conjunto de datos de prueba se descargó desde:

- Diputados. Proyecto de Ley con el Fondo Monetario Internacional / sesión extraordinaria del 08-Mar-2022: <https://youtu.be/O7M6qsPd0rI>
- Senadores - SESIÓN ESPECIAL 10-Ago-22: <https://youtu.be/8Jpwgw-kwQA>
- Cristina Fernández de Kirchner - Derecho de Defensa. Causa vialidad. 23-Ago-2022: <https://youtu.be/aNtbi1tSiWo>
- Cristina Fernández de Kirchner - Año 2000: <https://youtu.be/2t8h8E8-lkg>
- Fátima Florez imitando a Cristina Fernández de Kirchner - 05-Dic-2020: <https://youtu.be/I6t8goOfbXI>
- Se desestimó el set de datos de Mozilla (Datasets: Common Voices, 2022) ya que los registros antes mencionados proveen suficiente información para realizar este trabajo.

Conjunto de archivos de audio para Deep Fake

La última y más refinada de las pruebas, está orientada a detectar casos relacionados con Deep Fake (Fake You: Texto a Voz, 2022) (como ejemplo, desde la página web “FakeYou” se puede usar aprendizaje profundo para reproducir textos con la voz de algunos personajes famosos)

Para tales pruebas, se necesitan voces reales y sus correspondientes Fakes.

- Val Kilmer y su nueva voz generada con Inteligencia Artificial - 2022: <https://youtu.be/OSMue60Gg6s>

- Val Kilmer y su voz original en la película TOP GUN 1 (usada para predicción): <https://youtu.be/zq8rHTcveyA>
- Morgan Freeman y su voz generada con Inteligencia Artificial mediante Deep Fake en 2020: <https://youtu.be/F4G6GNFz0O8>
- Morgan Freeman, voz original: <https://youtu.be/TyyzL-AHPCA>
- Donald Trump, voz original: <https://youtu.be/gP-JkWgLPNU>
- Donald Trump, y su voz generada con Inteligencia Artificial mediante Deep Fake en 2020: <https://youtu.be/aPp5lcqgISk>

Preparación de ruido

Se realizó la carga de todas las muestras de ruido muestreadas a 16000 Hz.

Se dividieron las muestras de ruido en fragmentos de 1 segundo de duración cada una, a 16000 Hz.

Etapa del pre-procesamiento de registros de voz

Permite obtener un conjunto de datos que será procesado por un modelo de aprendizaje supervisado.

El objetivo es realizar predicciones con un alto nivel de confianza, para generar pruebas forenses donde el informe no resulte impugnado por motivos técnicos. En ese sentido, los atributos de la voz que componen el set de datos que se utilizará en el modelo, no deben ser distorsionados.

La identificación forense de hablantes por medio de métodos biométricos automáticos, implica que el método de predicción sea reproducible, utilizando muestras de voz sin alterar su cadena de custodia.

Este tópico se enfoca en brindar soporte a los métodos semiautomáticos y automáticos, que trabajan directamente con archivos de audio digitalizados, cuya conversión y normalización componen el conjunto de datos, siendo así el pre-procesado de los registros de voz una parte fundamental para crear éstos conjuntos de datos, que a la postre, serán utilizados como evidencia en casos donde se requiera validar si la voz de los registros pertenece al presunto implicado o no.

A continuación se desarrolla el proceso -paso a paso- para el pre-procesamiento de archivos de audio correspondientes a grabaciones de hablantes en procesos judiciales, que permitirá tanto a los peritos informáticos como a especialistas fonéticos forenses a reducir los tiempos de ejecución de pruebas y confección de dictámenes.

Repository - Preparación de datos

El primer conjunto de datos de prueba está disponible para descargar desde el repositorio público:

[https://www.kaggle.com/datasets/kongaevans/speaker-recognition-dataset?
resource=download](https://www.kaggle.com/datasets/kongaevans/speaker-recognition-dataset?resource=download)

Pre-procesamiento

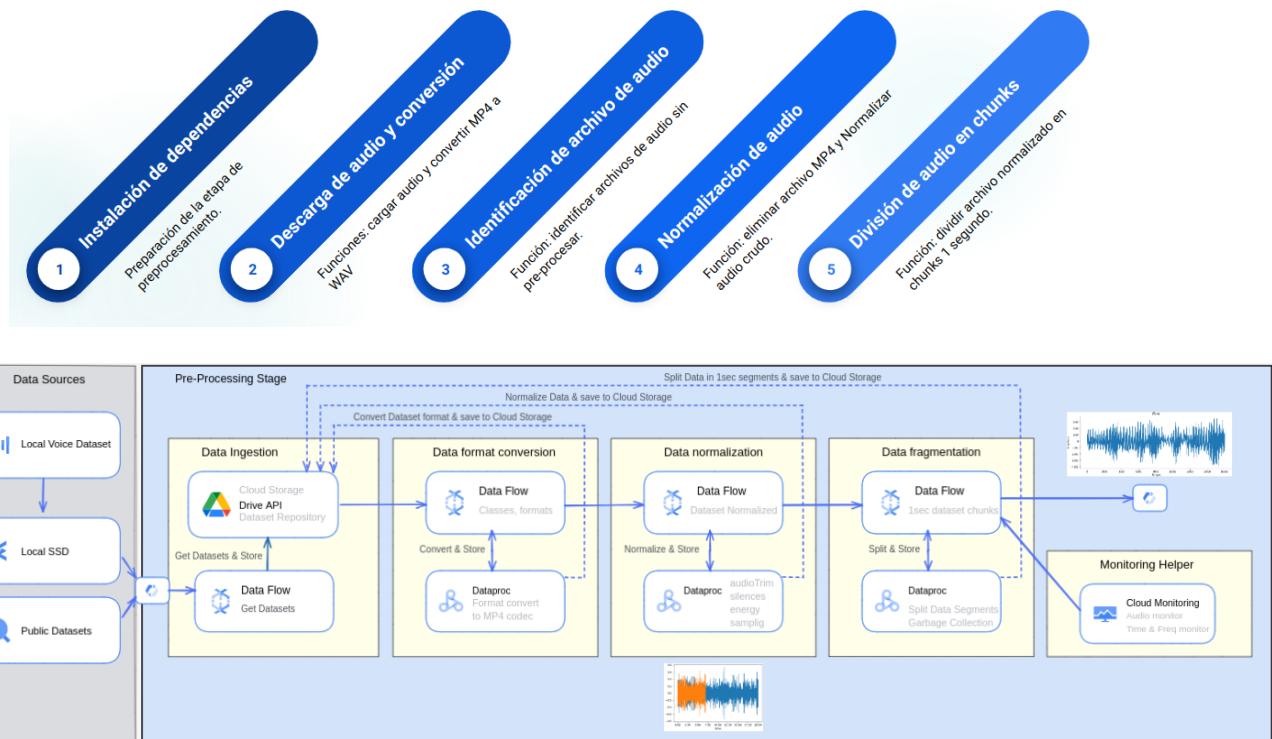


Ilustración 29. Etapas del proceso de pre-procesamiento

Preparación de la etapa de pre-procesamiento

Para aprovechar al máximo la capacidad de procesamiento al mínimo costo, el set de datos debe estar normalizado para llegar con mayor rapidez a un modelo entrenado en óptimas condiciones, de manera que la predicción se resuelva con un alto nivel de certeza. Para ello, es indispensable preparar el ambiente de la siguiente manera:

- Instalar dependencias que se usarán en el entorno Google-Colab.
- Importar las librerías.
- Inicialización y seteos.

En este caso, hay filas comentadas que fueron utilizadas para ejecutar el código en un entorno local sobre Anaconda (*Instalar Anaconda en Ubuntu*, 2021) y Jupyter notebook (*Jupyter Notebook*, 2021)

A la variable V_VIDEO_LINK se le asigna la ruta del video de YouTube.

A la variable V_LABELS DESDE EXCEL, se le asigna el nombre del archivo con los nombres de los hablantes, y el tiempo en segundos de inicio y fin donde se encuentre hablando dicha persona en el video de YouTube.

	A	B	C
1	seqs_ini	seqs_fin	Nombre_Senador
2	1857	2091	MariaBelenTapia
3	2125	2619	LuciaCorpacci
4	2633	2718	LuisPetcoffNaidenoff
5	2723	2763	PabloBlanco
6	2773	2911	CarolinaLosada
7	2921	3055	AnahelFernandezSanasti

Ilustración 30: Detalle del archivo de hablantes con referencia temporal (en segundos)

Función cargarAudio

Desde la URL de YouTube, descarga únicamente el audio en formato MP4.

Función convertirMP4aWAV

Convierte el formato de audio MP4 a archivo con formato WAV (Luis Pedro Coelho, Willi Richert, Matthieu Brucher, 2018)

Función identificarArchivoAudioSinPreProcesar

Devuelve la ruta donde se encuentra el archivo WAV sin preprocesar.

Función eliminarArchivoMP4

Permite eliminar el archivo original en formato MP4 para liberar espacio en el disco en la nube (unidad-drive del usuario que está ejecutando la notebook) (*os.path - Common pathname manipulations, 2022*)

- Secuencia de procesamiento del primer bloque principal (llamada a funciones)
- Bloque de lectura de archivo Excel con los datos de los hablantes

Usando la librería Pandas, se logró parametrizar el nombre de cada hablante, de manera que se considere como el nombre de cada etiqueta.

Además, para cada hablante, también se parametriza en el audio el tiempo desde que el hablante inicia su intervención hasta que finaliza.

Función normalizarAudioCrudo

Normaliza el audio a una potencia de -3dB (*Audio data augmentation, 2022*)
Esta operación no modifica los atributos del habla de la persona, sino que nivela la amplitud de la señal de audio para que sonidos suaves sean más fuertes, y sonidos muy fuertes sean suavizados.

Toma la muestra de audio completa y recorta sólo la porción de tiempo correspondiente al hablante que está procesando (*Denoising Data with FFT, 2020*)

Recorta la señal (*The SoX of Silence, 2009*) cuya amplitud no supere el 0.5% y su duración supere los 0.1 segundos, de manera que elimina silencios (*Audio Processing and Remove Silence using Python, 2020*)

Informa cuáles son los cambios que se realizaron a la señal de audio, y la guarda en disco en un archivo con el nombre de la etiqueta extraído del archivo Excel del bloque anterior.

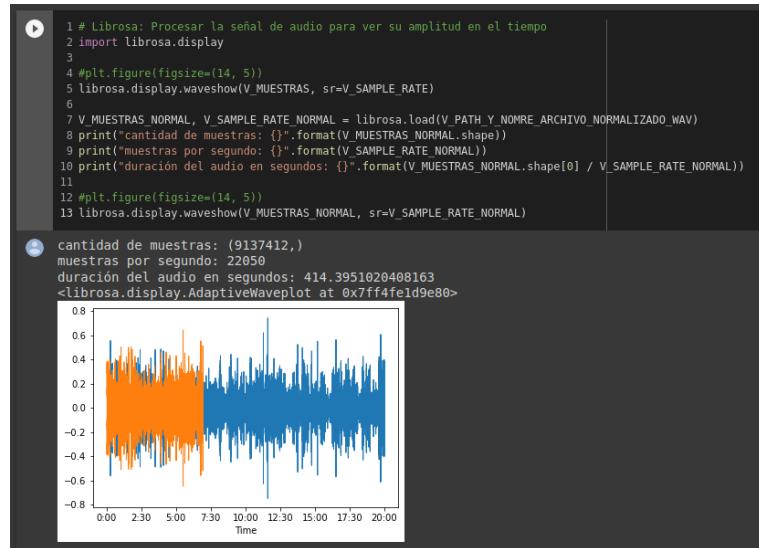


Ilustración 31: Forma de onda para audio sin preprocessar (azul) y audio preprocessado

(naranja)

Función dividirArchivoNormalizadoEnChuncks1seg

La función se encarga de dividir el archivo normalizado del paso anterior, en muestras de 1 segundo (*How to splice an audio file into 1 sec splices in python, 2021*)

Esto es esencial para poder aplicar transformadas rápidas de Fourier. Si no existe una carpeta con el nombre de la etiqueta, la crea.

Finalmente, mueve todos los archivos normalizados de 1 segundo, a la carpeta correspondiente a su etiqueta.

El resultado es un directorio (path)/nombre_de_hablante, que contiene múltiples archivos de 1 segundo normalizados, de dicho hablante.

Secuencia de procesamiento del segundo bloque principal

Este método automático permite obtener la muestra normalizada de los hablantes, para que el procesamiento de las mismas devuelva predicciones confiables.

Procesamiento

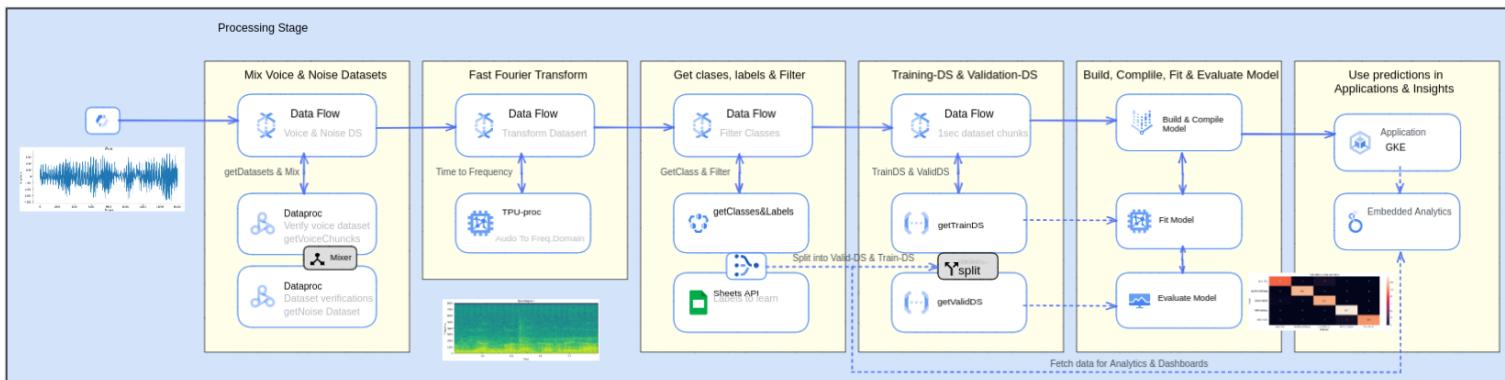


Ilustración 32: Etapas de procesamiento

Etapa del procesamiento

Configuración inicial

Es donde se importan las librerías de python necesarias para ejecutar el proceso (os, shutil, numpy, tensorflow, keras, pathlib, IPython) Además, se configuran los valores de las variables que serán utilizadas durante su ejecución.

Las muestras de audio están compuestas por 7 carpetas, divididas en 2 grupos: Muestras de voz, con 5 carpetas para 5 hablantes diferentes. Cada carpeta contiene 1500 archivos de audio, cada uno de 1 segundo de duración y muestreados a 16000 Hz.

Muestras de ruido de fondo, con 2 carpetas y un total de 6 archivos. Estos archivos duran más de 1 segundo (y originalmente no se muestrearon a 16000 Hz,

por eso se volverán a muestrear a 16000 Hz). Durante este proyecto, se usarán los 6 archivos de ruido, para crear 354 muestras de ruido de 1 segundo de duración que se usarán para el entrenamiento.

Las 2 categorías se ordenan en 2 carpetas. Audio: con todas las carpetas de muestra de voz por hablante y Ruido: con todas las muestras de ruido.

Preparación de ruido

Se realiza la carga de todas las muestras de ruido (muestreadas a 16000 Hz)

Se dividen las muestras de ruido en fragmentos de 1 segundo de duración cada una, a 16000 Hz.

Generación del conjunto de datos (DataSet)

El objetivo en esta etapa es construir un DataSet de audios con sus etiquetas. El bloque de código contiene las siguientes funciones de soporte:

path_to_audio: para leer y decodificar un archivo de audio.

add_noise: para mezclar un ruido sobre un registro de audio.

audio_to_fft: Permite obtener la primera mitad de la FFT, que representa las frecuencias positivas del registro de audio.

Aquí, los datos se dividen en 2 conjuntos: uno para entrenamiento y otro para validación. Al conjunto de entrenamiento, se lo mezcla con ruido y luego se obtiene la transformada de la señal de audio usando la función audio_to_fft.

Definición del modelo

En una función llamada “residual_block” permite alimentar una capa posterior a la siguiente capa de procesamiento, mientras entrenamos el modelo con la cantidad de capas que “efectivamente aprenden”

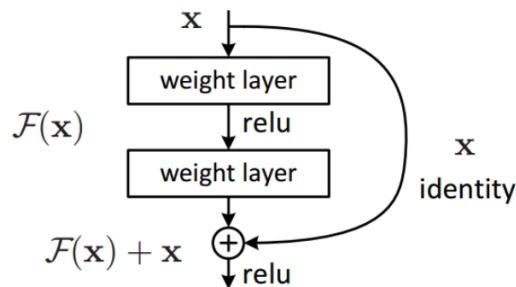


Ilustración 33: Función procesamiento residual con activación relu.

En otra función denominada “build_model” se especifica la función de activación = “RELU” (unidad rectificada lineal), ya que nuestro modelo no está limitado a valores -1, 0 ó 1.

En el bloque se define el algoritmo de “Adam” con tasa de aprendizaje por defecto (Adam es un método estocástico de gradiente descendente) y la función de pérdida “sparse_categorical_crossentropy” (para medir la disimilitud entre la distribución de las etiquetas de clase observadas y las probabilidades previstas de pertenencia a una clase)

La función utilizada como métrica del rendimiento del modelo es “accuracy” (la relación entre el número de predicciones correctas y el número total de muestras de entrada)

Finalmente, se agregó el método “EarlyStopping” para monitorear el rendimiento y detener el proceso de aprendizaje una vez alcanzado el umbral deseado.

Entrenamiento

Consiste en ejecutar la función de entrenamiento “model.fit”. En esta etapa comienza el aprendizaje.

Nota: Se debería ejecutar con TensorFlow 2.3 o superior, ya que usando menos recursos de hardware, la experiencia de ejecución de aprendizaje podría demorar días, contra minutos de procesamiento en el caso de TensorFlow.

Como alternativa y aun sin costo para evaluar este prototipo -dentro de los límites y políticas de Google- (*Elige el plan de Colab adecuado para ti*, 2022), se sugiere usar GPU en Google Colab.

Evaluación

Siguiendo los pasos indicados para ejecutar el modelo, la métrica de evaluación -en una de las pruebas- devolvió:

loss: 0.1474 - accuracy: 0.9423

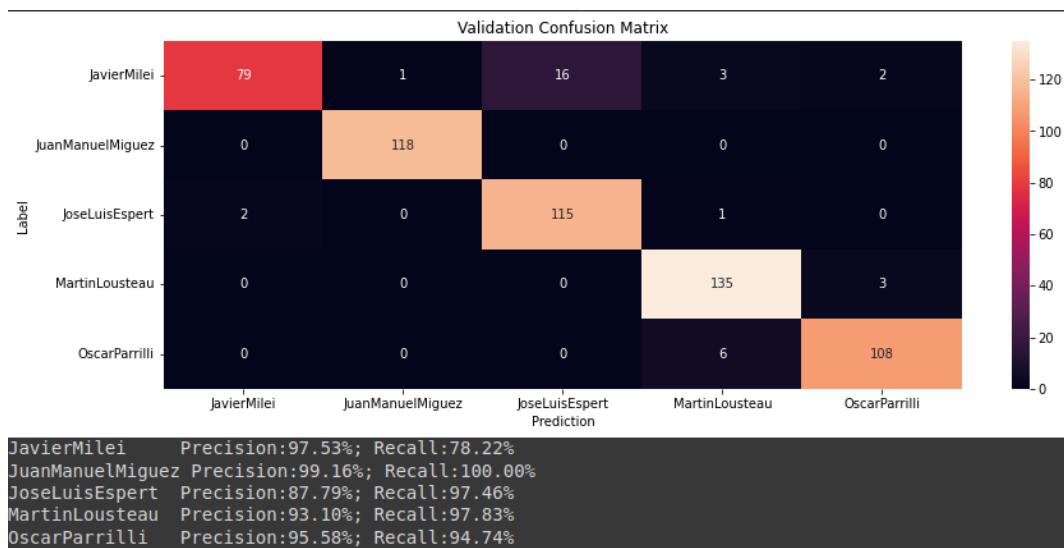


Ilustración 34: métricas de una ejecución del prototipo con datos de prueba

En este caso, se observó que en la fila 1 de la matriz de confusión, devolvió Recall < 0,8. Esto podría representar errores en la predicción de la clase; sin embargo, como Recall para esa clase está por encima de 0,7 , entonces se asume que el modelo está bien entrenado.

Demostración

En esta etapa del proyecto, la ejecución del bloque toma de manera aleatoria alguna muestra de audio existente, la mezcla con un nuevo ruido, y de esta manera se simula el registro de voz de un “hablante”. Finalmente se somete a clasificación. Lo esperado es que el modelo realice una predicción dentro del intervalo de confianza demostrado previamente.

En consecuencia, el modelo va a predecir a qué hablante corresponde la voz, y además, va a informar de qué hablante efectivamente se trata (predicción vs. realidad). Para ilustrar aún más, se provee un reproductor que permite En esta etapa del proyecto, la ejecución del bloque toma de manera aleatoria alguna muestra de audio existente, la mezcla con un nuevo ruido, y de esta manera se simula el registro

de voz de un “hablante”. Finalmente se somete a clasificación. Lo esperado es que el modelo realice una predicción dentro del intervalo de confianza demostrado previamente.

En consecuencia, el modelo va a predecir a qué hablante corresponde la voz, y además, va a informar de qué hablante efectivamente se trata (predicción vs. realidad). Para ilustrar aún más, se provee un reproductor que permite escuchar la pista de audio seleccionada como hablante para poder contrastar el resultado utilizando el método fonético-acústico (empírico)

Mejoras de desempeño

Rendimiento

Uno de los objetivos alcanzados, fue la optimización del tiempo de entrenamiento y validación. En primera instancia, tomando el conjunto de 140 hablantes, con procesamiento local y Hardware / Software mencionados, para VALID_SPLIT=0,1, el tiempo de cada EPOCH fue 10702 segundos, equivalente a 2.97 horas. Esto quiere decir, que con este equipo de procesamiento, 3 EPOCHS tardaron 30024 segundos, equivalente a 8.34 horas. Luego, el accuracy fue muy bajo <10%, dando por descartado este tipo de prueba.

En iguales condiciones, pero en Google Colab, con GPU, cada EPOCH tardó 5:36 horas, y se interrumpió el procesamiento.

En iguales condiciones, para TPU, el tiempo de cada EPOCH fue de 0,38 horas. Sin embargo, el accuracy para 3 EPOCHS no fue aceptable <10%.

El tiempo de procesamiento debe finalizar antes de las 12 horas, ya que es el tiempo límite fijado en Colab para mantener la sesión activa. Fuera de ese tiempo, el entrenamiento se debe repetir. Por eso, a 30 minutos por EPOCH, 10 EPOCHS equivalen a 300 minutos (o 50 horas, que no son aceptables).

Usando VALID_SPLIT = 0.8 y EPOCHS = 10 con TPU, el procesamiento de 1 EPOCH demoró hasta 11 minutos. 11 minutos x 10 EPOCHS = 110 minutos = 1,8 horas. Sin embargo, VALID_SPLIT = 0,8 implica menor cantidad de archivos de entrenamiento, pero mayor cantidad de archivos para validación. En este otro caso extremo, la validación demoró más de 1 hora y el proceso se finalizó intencionalmente por no ser viable.

Menor cantidad de hablantes

Con 45 hablantes (en lugar de 140), VALID_SPLIT = 0.1 y EPOCHS = 5, se estimó estimó 1 EPOCH = 18:50 minutos y 5 EPOCHS ~ 1,57 horas, siguiendo estos resultados:

- en 1 EPOCH se alcanzó ~ 0.48 accuracy
- en 2 EPOCHS se alcanzó ~ 0.54 accuracy
- en 3 EPOCHS se alcanzó ~ 0.56 accuracy
- en 4 EPOCHS se alcanzó ~ 0.64 accuracy - 19 minutos promedio por EPOCH
- en 5 EPOCHS se alcanzó ~ 0.65 accuracy

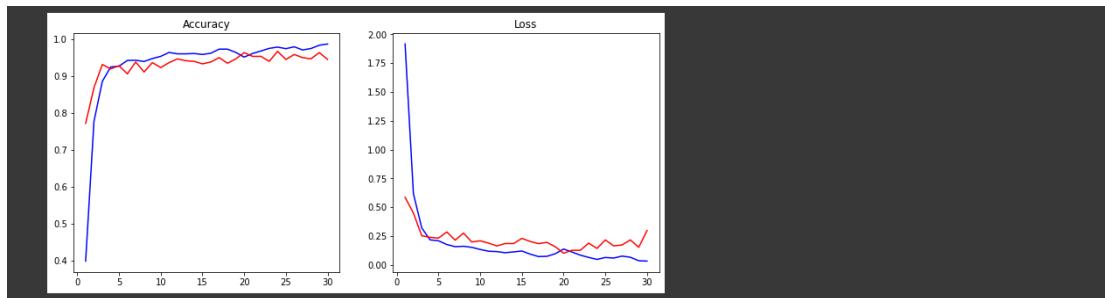


Ilustración 35: Valor de métrica Accuracy vs. EPOCHS (a la izquierda) y Valor de métrica Loss vs. EPOCHS (a la derecha)

En consecuencia, ciertas voces que se usaron en el entrenamiento, fueron reconocidas casi con certeza. Las voces que no se usaron para entrenar, no fueron reconocidas. Algunas voces que fueron entrenadas, no se reconocieron con certeza (Errores tipo I y II)

Luego de realizar varios ajustes a modo de prueba y error, empíricamente se identificó que 5 hablantes con 500 archivos en promedio por hablante, son necesarios para conseguir que el modelo consiga realizar predicciones aceptables.

Con VALID_SPLIT = 0.1 y EPOCHS = 50, promedio 217 segundos o 3,6 minutos por EPOCH y Accuracy ~ 0.95, Precision y Recall > 0,90. Todas las voces que se usaron en el entrenamiento, fueron reconocidas casi con certeza.

Las voces que no se usaron para entrenar, no fueron reconocidas.

En una nueva prueba, TPU, VALID_SPLIT = 0.1, EPOCHS = 50 \Rightarrow alcancé modelo estable en 16 EPOCHS, en 3448 segs (57.5 minutos) con loss: 0.1525 - accuracy: 0.9494.

Durante las pruebas de performance y optimización, se utilizaron conjuntos de datos con menos de 50 archivos por hablante para validar la efectividad frente a Deep Fake. Para los casos de los hablantes Val Kilmer y Morgan Freeman, esta cantidad de archivos provocó un sobre-entrenamiento del modelo, devolviendo predicciones absurdas.

Optimizando VALID_SPLIT de 0,1 a 0,9

NOTEBOOK	VALID_SPLIT	EPOCHS	USED_EPOCHS	ELAPSED_TIME	MEAN_EPOCH_TIME	accuracy	loss	Backend
procesamiento_V0020101.ipynb	0.1	50	16	3448	215.5	0.9494	0.1525	TPU
procesamiento_V0020103.ipynb	0.2	50	16	1825	114.1	0.9378	0.1984	GPU
procesamiento_V0020110.ipynb	0.25	50	27	5502	203.7	0.9394	0.1714	-
procesamiento_V0020104.ipynb	0.3	50	17	967	56.8	0.9356	0.1806	GPU
procesamiento_V0020111.ipynb	0.35	50	17	3225	189.7	0.9269	0.2204	TPU
procesamiento_V0020105.ipynb	0.4	50	22	3760	62.7	0.9309	0.2062	TPU
procesamiento_V0020102.ipynb	0.5	50	37	4532	122.5	0.9483	0.1667	TPU
procesamiento_V0020106.ipynb	0.6	50	36	2463	68.4	0.9307	0.2040	GPU
procesamiento_V0020107.ipynb	0.7	50	33	4162	126.1	0.9232	0.2493	-
procesamiento_V0020108.ipynb	0.8	50	20	1838	91.9	0.8288	0.4750	-
procesamiento_V0020109.ipynb	0.9	50	35	3637	103.9	0.8769	0.4366	-

Luego de haber ejecutado diferentes pruebas de carga, se pudo observar resultados óptimos con GPU ; VALID_SPLIT=0,2 ; 5 hablantes con 500 archivos de

audio en formato WAV, de 1 segundo en promedio, con muestreo a 16000 Hz (1 canal, mono) -3dB de potencia, nivelado y sin silencios.

⇒ El modelo quedó estable en 16 EPOCHS, en 1825 segs (30.41 minutos) con loss: 0.1984 - accuracy: 0.94; Precision >0,95 y Recall > 0,94.

Resultados

Reglas Generales / Restricciones / Interpretación

- ✓ No es posible entrenar con la voz de un sólo hablante y predecir compatibilidad con otros hablantes diferentes, ya que es un modelo supervisado.
- ✓ Si el modelo fue entrenado con un lote de n -hablantes y ninguno de ellos corresponde a la voz viva que necesito predecir, entonces la predicción será del orden igual o inferior a 98% de accuracy.
- ✓ Si el modelo fue entrenado con un lote de n -hablantes y alguna de las voces corresponde a la voz viva a predecir, entonces la predicción será acertada, y del orden superior a 98% de accuracy.
- ✓ Si el modelo fue entrenado con un lote de n -hablantes y alguna de las voces corresponde a la voz a predecir, pero la predicción devolvió métricas entre 95% y 98% de accuracy, entonces será conveniente resolver el problema de compatibilidad complementando el análisis con otro método de los ya mencionados para desambiguar resultados.
- ✓ Con métrica accuracy por debajo del 95%, no se podrá predecir compatibilidad entre la voz a predecir y el lote de datos que entrenó al modelo.
- ✓ Ante Deep Fake, con un modelo bien entrenado, el modelo falla.
- ✓ El modelo detecta bien a imitadores que son personas físicas (no Deep Fake)
- ✓ Si la voz viva a predecir no coincide temporalmente (dentro de un intervalo de a lo sumo: 2 años) con la muestra de voz de la misma persona atemporal, entonces, la predicción va a fallar.

- ✓ Los trastornos funcionales de la voz no alteran el resultado de la predicción.
- ✓ El modelo no se logró probar con trastornos orgánicos de la voz.
- ✓ Para minimizar los falsos positivos, es conveniente entrenar el modelo con un set de datos que tenga una cantidad balanceada de archivos de audio para cada hablante.
 - ✓ Las mismas son reproducibles, es decir, se pudo repetir el experimento en distintos entornos y alcanzar el mismo resultado independientemente de la plataforma y sesión.

Performance

- El entrenamiento óptimo se logra con:
 - Muestras de audio
 - formato WAV
 - muestreo a 16kHz, 1 canal (mono)
 - no menos de 500 archivos por hablante
 - sin silencios
 - -3dB de potencia de referencia
 - 1 segundo por archivo de muestra
 - Parámetros
 - VALID_SPLIT = 0.2
 - SAMPLING_RATE = 16000

- EPOCHS = 50
 - HW y SW
 - Cloud, con Google Colab, community edition (sin abono)
 - 100GB de espacio de disco en la nube
 - 12GB de memoria RAM
 - procesamiento GPU (1 thread)

Entrenamiento inicial con 1 voz

- Si el modelo se entrena con 1 voz, la predicción no es confiable.
- Si el modelo se entrena con más de 1 voz, el modelo predice a cuál de las voces entrenadas se acerca más la voz viva (incógnita)

Imitadores

- Si el modelo se entrena con voces de una persona y de su imitador, el modelo predice correctamente cuando la Voz Viva se trata del imitador.
- Si el modelo se entrena con voces de una persona, pero no con la de su imitador, el modelo predice compatibilidad menor al 90% con la Voz Viva de su imitador.

Mismo hablante, pero difieren 20 años.

- Si el modelo se entrena con voces de una persona, y la Voz Viva corresponde al mismo hablante, con diferencia de 20 años, entonces el modelo interpreta que la Voz Viva no es compatible con la voz del hablante que entrenó el modelo.

Entrenamiento con set de 5 hablantes (performante)

- Si el modelo se entrena con voces de género femenino, y la Voz Viva es de género masculino, el modelo predice compatibilidad menor al 80% con alguna categoría de las entrenadas.
- Si el modelo se entrena con voces femeninas, y la Voz Viva es femenina pero no forma parte del set de entrenamiento, entonces el modelo predice compatibilidad menor al 90% con alguna categoría de las entrenadas.
- Si el modelo se entrena con voces femeninas, y la Voz Viva corresponde a una persona que se usó para entrenar el modelo, entonces la predicción es mayor al 98%.
- Si el modelo se entrena con voces masculinas, vale todo lo anterior.

Entrenamiento con hablante y su Deep Fake

- Si el modelo se entrena con un hablante que sea fácil obtener su muestra Deep Fake, y otras 3 voces de diferentes hablantes en su mismo idioma; y la Voz Viva es el Deep Fake obtenido antes. Entonces el Deep Fake tiene una predicción mayor al 98%, compatible con la humana.

Entrenamiento con trastorno orgánico de la voz

- Si el modelo se entrena con 3 hablantes, y otro hablante del cual también se obtuvo la muestra de voz con disfonía, y la Voz Viva es la voz con disfonía obtenida antes, entonces el modelo predice compatibilidad >99% con la voz del hablante correcto sin disfonía.

CAPÍTULO 5: CONCLUSIONES

Como ocurrió en el año 1891 con el método Dactiloscópico o en año 1983 con el método genético del ADN, en cuanto a sus tardías implementaciones, a fines del año 2017 expertos en Lingüística, Fonoaudiología, Estadistas y también Jueces, aún evitaban los enfoques de reconocimiento automático de hablantes y en su lugar implementaban los métodos tradicionales combinados.

Conforme evolucionaron el poder de procesamiento computacional y las técnicas de aprendizaje profundo, se alcanzó un hito difícil de ignorar y que en este informe se logró demostrar: **un desarrollo basado en una de las ramas de "Inteligencia Artificial", conocida como "Aprendizaje Profundo" (Deep Learning por su traducción al inglés) logró identificar hablantes automáticamente, con métricas que hasta el momento eran alcanzadas por los métodos antes mencionados (huella dactilar y ADN)**

Para lograr dicho propósito, en esta Tesis se desarrolló el pre-procesamiento de todas las muestras de audio para obtener un conjunto de datos de entrenamiento y validación para el modelo, mediante: la conversión de su formato, normalización de energía y silencios, división de señal en muestras de 1 segundo basado en la transformación de señales en el dominio del tiempo, a muchas señales de corta duración consideradas “*periódicas*” en el dominio de la frecuencia mediante **Transformadas Rápidas de Fourier** y la mezcla de audio con ruido.

Luego, se desarrolló el procesamiento que consiste en un método supervisado que implementa una red neuronal convolucional de 1 dimensión (CNN-1D) con función de activación RELU, algoritmo de Adam, función de pérdida

"sparse_categorical_crossentropy", y método de monitoreo/corte "EarlyStopping", con métrica "accuracy" para alcanzar resultados evaluados mediante matriz de confusión, y métricas de calidad.

Todo este esfuerzo fue desplegado en la nube, utilizando la herramienta Google Colab para ejecutar código en lenguaje Python e integrado con el espacio de almacenamiento en la nube (Drive) - Cabe destacar que el código fuente se puede ejecutar gratuitamente, utilizando procesador GPU y como el proyecto pasó por una etapa de ajuste de rendimiento, también se puede ejecutar con procesador TPU.

Esta prueba forense le otorga al perito informático, la facultad de emitir un dictamen, aceptando o rechazando la hipótesis: "***La muestra dubitada de voz del hablante X es compatible con una exactitud del **,***% con la muestra indubitada de voz del hablante Y***". Luego, el juez o el jurado podrán usar ese dictamen con el tenor y la fuerza de evidencia y agregar toda información a priori que no dependa de la misma para su deliberación y sentencia final.

Habiendo conservando la prueba con la debida cadena de custodia, bajo las mismas condiciones de ejecución, los resultados expresados en métricas de exactitud que avalan esta tesis podrán ser repetidos las veces que sean necesarias, consiguiendo el mismo resultado (minimizando el riesgo de que alguna de las partes solicite impugnar el mismo por controversias técnicas que deberá argumentar)

Con estos lineamientos, el perito habrá contemplado el veredicto del caso Daubert (Daubert, 2016) y estará en condiciones de cumplir su rol de oficial de justicia cuando los puntos de pericia estén fuertemente vinculados con esta tesis.

Por otro lado, quedó demostrado que este método no resuelve Deep Fakes.

Futuras líneas de investigación

Someter el modelo a pruebas de certificación y ensayos clínicos.

Fase 1

Suponer este ensayo como una prueba clínica de Fase 1, ya que hasta el momento tiene éxito en escenarios de *Laboratorio*.

Fase 2

Debería contemplar un universo de hablantes mayor, y mayor poder de procesamiento para someter el modelo a pruebas categóricas sobre los casos que en Fase 1 fueron exitosos.

Fase 3

Debería someter el modelo a validación de expertos en todas las áreas de incumbencia en la identificación de hablantes no automática.

Autenticación de identidad de aplicaciones

Este frente requiere mayor investigación, ya que quedó demostrado que DeepFake puede vulnerar la identificación automática de hablantes.

Diagnóstico automático de afecciones cardiológicas

Las señales analógicas periódicas (ejemplo: electrocardiogramas) se pueden usar para entrenar este modelo y obtener una predicción acertada. Tener especial cuidado con los errores del tipo II. Los falsos negativos deben ser lo mínimo posible.

Evolución en el tiempo de la misma persona

Analizar audios de la misma persona, proponiendo un límite hacia atrás en el tiempo (cuánto atrás puedo ir) para poder identificar hablantes de manera automática.

CAPÍTULO 6: BIBLIOGRAFÍA

Citas Bibliográficas

Transcripción de la solicitud de generación de prueba y la designación de un experto:

“CIV 034762/2017, ARIAS, DIEGO ANTONIO C/ ARIAS DE PIÑEIRO,
MARTA MAFALDA Y OTRO S/RENDICION DE CUENTAS, a fs.1105,
C-PERITO INGENIERO EN INFORMÁTICA : Se designe por este juzgado
y secretaría Perito Ingeniero en informática, quien previa aceptación del
cargo por ante el actuario, deberá citar a Pablo Andrés Piñeiro, a fin de
"auditar" el dvd que se agrega con la " llamada que le hiciera a Leonor
Coca" y determinar si la voz le pertenece al co demandado Piñeiro. Se
realizarán las pericias en presencia de la parte actora, con citación de la
misma conforme el art. 471 del Código Procesal.” - <https://www.pjn.gov.ar>

Referencias Bibliográficas

- David L. Faigman, David H. Kaye, Michael J. Saks, Joseph Sanders. (1997) *Modern scientific evidence: The law and science of expert testimony* (pp. 335-346)
- J. Gibbons and M. T. Turell (Eds.) (2008) *Dimensions of forensic linguistics Amsterdam/Philadelphia* John Benjamins Publishing.
- P. Rose. (2002) *Forensic Speaker Identification* London: Taylor & Francis.
- Jorge Gurlekian. (2016) *Laboratorio de Investigaciones Sensoriales (LIS) - Primer simposio nacional de ciencia y justicia - La identificación forense de voces sustentada en bases de datos locales, regionales e internacionales*
<https://www.conicet.gov.ar/wp-content/uploads/GURLEKIAN.pdf>
- Pedro Univaso. (2016) *Towards a Unified Methodology for Forensic Speaker Identification.* (pp. 1-6)
- H. F. Hollien, (2002) *Forensic voice identification.* Academic Press.
- International Association for Forensic Phonetics and Acoustics (2007) *Resolution on voiceprints* <http://www.iafpa.net/voiceprintsres.htm>
- Augspach, F. (2003) *Disfonías profesionales su evaluación y tratamiento. 1º parte.* Revista Fonoaudiológica, Buenos Aires. (pp. 81- 88)
- Molina, N. (2002) *Reflujo y trastornos de la voz: un abordaje interdisciplinario, Fonoaudiologica.* ASALFA, Buenos Aires, Tomo 48 #1 (pp. 7)
- Chin-Hui Lee, Frank K. Soong, Kuldip K. Paliwal. (1996) *Automatic Speech and Speaker Recognition.* AT&T Bell Laboratories.

S. Jha et al. (2019) *Deep Learning Approach for Software Maintainability Metrics Prediction* in IEEE Access, vol. 7, (pp. 61840-61855)

B. Miller. (1997) *Everything you need to know about automated biometric identification*. Security Technol. Design.

R. Chandrasekaran. (1997) *Brave new whorl: ID systems using the human body are here, but privacy Issues persist*. Washington Post,

A. Davis. (1997) *The body as password, wired*.

F. James. (1997) *Body scans could make ID process truly personal*. Chicago Tribune.

J.R.Hilera, V.J. Martínez. (1995) *Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones*. Addison-Wesley Iberoamericana.

Luis Pedro Coelho, Willi Richert, Matthieu Brucher. (2018) *Building Machine*

Learning Systems with Python. (pp.268) Retrieved from

<https://books.google.com.ar/books?id=owZnDwAAQBAJ&pg=PA268&lpg=PA268&dq=run+sox+command+from+notebook&source=bl&ots=T40JBzc17T&sig=ACfU3U1znnzvQixxO14Oj->

NT8jidjTe27g&hl=es&sa=X&ved=2ahUKEwjajcOQ_rX5AhXQkZUCHa06C_wQ6AF6BAg9EAM#v=onepage&q=run%20sox%20command%20from%20notebook&f=false

Jason Brownlee. (2016) *Machine Learning Mastery with Python*.

Documentos Electrónicos

El caso Daubert. (2016). Retrieved from

[https://www.academia.edu/28305930/LA PRUEBA PERICIAL EN LA EXPERIENCIA ESTADOUNIDENSE EL CASO DAUBERT](https://www.academia.edu/28305930/LA_PRUEBA_PERICIAL_EN_LA_EXPERIENCIA_ESTADOUNIDENSE_EL_CASO_DAUBERT)

Diario Judicial. (2010). Retrieved from

<https://www.diariojudicial.com/nota/13402#:~:text=De%20esta%20manera%2C%20existen%20actualmente,31.945%20hombres%20y%2026.962%20mujeres>

Elecciones Colegio de abogados. (2022). Retrieved from

<https://www.cronista.com/economia-politica/votan-los-abogados-portenos-cuando-es-la-eleccion-quienes-pueden-participar-y-que-listas-se-presentan/>

Poder Judicial de la Nación Argentina. (2022). Retrieved from <https://consultaperitos.pjn.gov.ar>

Softjourn. (2020). <https://softjourn.com/insights/security-considerations-in-voice-authentication#:~:text=Voice%20authentication%20is%20a%20biometric,their%20voices%20themselves%20as%20>

Extract audio from YouTube video using Python. (2022). Retrieved from

<https://blog.balasundar.com/extract-audio-from-youtube-video-using-python> y https://github.com/SBalaSundar/youtube_audio_extractor

PyTube3. (2022). Retrieved from

<https://pypi.org/project/pytube3/> <https://github.com/YouTubeDownload/YouTubeDownload>

SoX - Sound eXchange. (2015). Retrieved from <http://sox.sourceforge.net/>

Piping SoX in Python. (2022). Retrieved from <https://pypi.org/project/sox/>

Pysox: Leveraging the audio signal processing power of sox in Python. (2016).

Retrieved from

<https://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/08/bittner-pysox.pdf>

pysox Documentation. (2021). Retrieved from

<https://buildmedia.readthedocs.org/media/pdf/pysox/latest/pysox.pdf>

Source code for sox.transform. (2016). Retrieved from

https://pysox.readthedocs.io/en/latest/_modules/sox/transform.html

Pysox: Python Wrapper Around Sox. (2022). Retrieved from

<https://morioh.com/p/8ac4dd1bbbcc>

Sox and ffmpeg.ipynb. (2022). Retrieved from

https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/sox_and_ffmpeg.ipynb#scrollTo=3TkyJuXBmD_M

Audio Handling Basics. (2020). Retrieved from <https://hackernoon.com/audio-handling-basics-how-to-process-audio-files-using-python-cli-jo283u3y>

Pydub. (2022). Retrieved from <http://pydub.com/>

A New Silence Removal and Endpoint Detection Algorithm for Using Audio in Ipython. (2022). Retrieved from

https://notebook.community/imsparshtanford-mir/notebooks/ipython_audio#Playing-Audio

pyAudioAnalysis. (2015). Retrieved from

https://www.researchgate.net/publication/286637817_pyAudioAnalysis_An_Open-Source_Python_Library_for_Audio_Signal_Analysis

Python Program - Plot A Wave Audio File. (2020). Retrieved from

<https://www.youtube.com/watch?v=4rzpMA6CUPg>

Basic Sound Processing in Python. (2015). Retrieved from

<https://www.youtube.com/watch?v=0ALKGR0I5MA> ,
https://nbviewer.org/github/AllenDowney/ThinkDSP/blob/master/code/scipy2015_demo.ipynb , <https://tinyurl.com/scipy15dsp>

Using Audio in Ipython. (2022). Retrieved from

https://notebook.community/imspars/stanford-mir/notebooks/ipython_audio ,
https://notebook.community/imspars/stanford-mir/notebooks/ipython_audio#Playing-Audio

Essentia.standard Python tutorial. (2022). Retrieved from

https://notebook.community/ChristianFrison/essentia/src/examples/tutorial/essentia_tutorial_standard

Essentia Python tutorial. (2022). Retrieved from

https://github.com/MTG/essentia/blob/master/src/examples/python/essentia_python_tutorial.ipynb ,
https://notebook.community/ChristianFrison/essentia/src/examples/tutorial/essentia_tutorial_standard

Audio data augmentation. (2022). Retrieved from

https://pytorch.org/audio/main/tutorials/audio_data_augmentation_tutorial.html,

Instalar Anaconda en Ubuntu. (2021). Retrieved from

<https://ourcodeworld.co/articulos/leer/1609/como-instalar-anaconda-en-ubuntu-2004>

Anaconda Distribution. (2022). Retrieved from

<https://www.anaconda.com/products/distribution>

Anaconda Documentation. (2022). Retrieved from

<https://docs.anaconda.com/anaconda/install/verify-install/#nav-problems>

How to set Jupyter Notebook to open on browser automatically. (2021). Retrieved

from <https://stackoverflow.com/questions/55756151/how-to-set-jupyter-notebook-to-open-on-browser-automatically>

os.path - Common pathname manipulations (2022). Retrieved from

<https://docs.python.org/3/library/os.path.html>

Denoising Data with FFT. (2020). Retrieved from <https://www.youtube.com/watch?v=s2K1JfNR7Sc> , <https://www.youtube.com/watch?v=spUNpyF58BY>

How to splice an audio file into 1 sec splices in python. (2021). Retrieved from

<https://stackoverflow.com/questions/36799902/how-to-splice-an-audio-file-wav-format-into-1-sec-splices-in-python?rq=1>

Audio Processing and Remove Silence using Python. (2020). Retrieved from
<https://ngbala6.medium.com/audio-processing-and-remove-silence-using-python-a7fe1552007a>

The SoX of Silence. (2009). Retrieved from
<https://digitalcardboard.com/blog/2009/08/25/the-sox-of-silence/>

Audio Processing and Remove Silence using Python. (2020). Retrieved from
<https://ngbala6.medium.com/audio-processing-and-remove-silence-using-python-a7fe1552007a>

Remove silents using VAD. (2020). Retrieved from <https://malaya-speech.readthedocs.io/en/latest/remove-silent-vad.html>

Mozilla lanza el segundo mayor conjunto de datos de voz público. (2017). Retrieved from
<https://blog.mozilla.org/press-es/2017/11/29/commonvoicemozilladatosvoz/>

Datasets: Common Voices. (2022). Retrieved from
<https://commonvoice.mozilla.org/es/datasets>

Consejo de Profesionales de Ciencias Informáticas de la Ciudad de Buenos Aires. (2022). Retrieved from <https://www.cpci.org.ar/>

Forensia: un sistema de identificación forense por voz. (2020). Retrieved from
https://www.researchgate.net/publication/344929766_FORENSIA_un_sistema_de_identificacion_forense_por_voz

Obtención de sonogramas con los formantes realizados usando el método de predicción lineal. (1999). Retrieved from

https://www.ub.edu/journalofexperimentalphonetics/pdf-articles/EFE-X-Jbobadilla_Pgomez_Jbernal-Obtencio_de_sonogramas_con_formantes_realizados_usando_LPC.pdf

Caracterización de los indicadores acústicos de la voz. (2008). Retrieved from
<https://www.redalyc.org/pdf/874/87411107005.pdf>

Acoustic Measures and Self-reports of Vocal Fatigue by Female Teachers. (2006). Retrieved from
[https://www.jvoice.org/article/S0892-1997\(06\)00133-0/fulltext](https://www.jvoice.org/article/S0892-1997(06)00133-0/fulltext)

Análisis acústico de la voz normal y patológica: Anagraf y Praat. (2012). Retrieved from
<https://www.redalyc.org/pdf/180/18026361002.pdf>

Transformada de Fourier – Matlab. (2016). Retrieved from
http://www.sc.ehu.es/sbweb/fisica3/simbolico/fourier/fourier_1.html

La Transformada de Fourier – definición y propiedades. (2020). Retrieved from
<https://dademuch.com/2020/02/05/la-transformada-de-fourier-definicion/>

FFT, Transformada Rápida de Fourier. (2013). Retrieved from
<http://lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-Schmidt%20Ana%20Luc%C3%A9Da.pdf>

How to use metrics for Deep Learning with Keras in Python. (2017). Retrieved from
<https://machinelearningmastery.com/custom-metrics-deep-learning-keras-python/>

Keras – Metrics. (2022). Retrieved from <https://keras.io/api/metrics/>

Keras 2 release notes. (2020). Retrieved from

<https://github.com/keras-team/keras/wiki/Keras-2.0-release-notes>

Keras-metrics 1.1.0. (2020). Retrieved from <https://pypi.org/project/keras-metrics/>

English speaker accent recognition using Transfer Learning. (2022). Retrieved from

https://keras.io/examples/audio/uk_ireland_accent_recognition/

Fake You. Texto a Voz. (2022). Retrieved from <https://fakeyou.com/>

Elige el plan de Colab adecuado para ti. (2022). Retrieved from

[https://colab.research.google.com/signup/pricing?
utm_source=resource_tab&utm_medium=link&utm_campaign=payg_learn](https://colab.research.google.com/signup/pricing?utm_source=resource_tab&utm_medium=link&utm_campaign=payg_learn)

[more](#)

Marisol Hernández López. *Sistema para reconocimiento de hablantes dependiente e*

independiente del texto. (2004). Retrieved from

<https://tesis.ipn.mx/jspui/bitstream/123456789/1592/1/marisol.pdf>

Yumoto et al, (1982). Retrieved from

<https://asa.scitation.org/doi/abs/10.1121/1.387808>

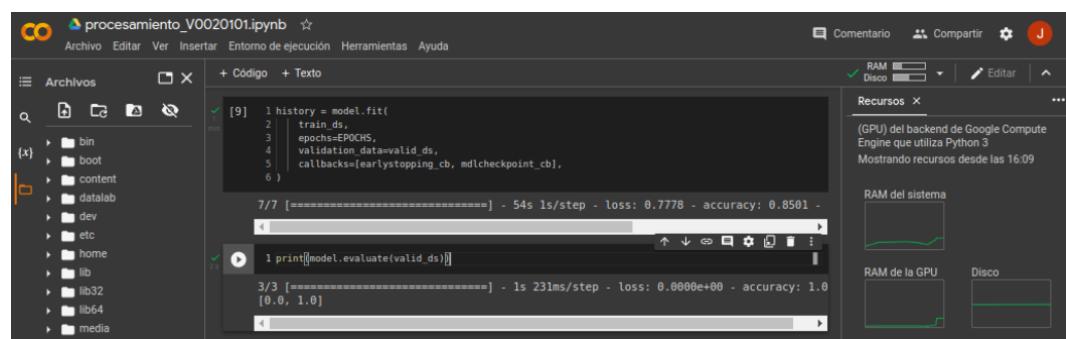
CAPÍTULO 7: APÉNDICES

Ejecución de pruebas

Prueba 1 – entrenamiento con 1 hablante

El modelo es entrenado únicamente con la voz de un hablante.

La predicción se calcula a partir de una voz distinta a la única voz que entrenó el modelo.



```
[9] 1 history = model.fit(  
2     train_ds,  
3     epochs=EPOCHS,  
4     validation_data=valid_ds,  
5     callbacks=[earlystopping_cb, mdlcheckpoint_cb],  
6 )  
7/7 [=====] - 54s 1s/step - loss: 0.7778 - accuracy: 0.8501 -  
1 print(model.evaluate(valid_ds))  
3/3 [=====] - 1s 231ms/step - loss: 0.0000e+00 - accuracy: 1.0  
[0.0, 1.0]
```

Ilustración 39: métricas para la prueba 1.

- Notebook = voiceRecognition_v0020101
- Audios para entrenamiento = prueba_001.xls
- nombres_políticos = [CristinaFernandezDeKirchner]
- voz_viva = cualquiera, distinta a la del hablante entrenado.

Con 1 EPOCH, en Google colab, con GPU, accuracy=1

El aprendizaje de las muestras demoró 54 segundos, para 945 archivos de 1 segundo.

```

8 valid_audio_paths_vivo = []
9 valid_labels += []
10 test_ds = paths_and_labels_to_dataset(valid_audio_paths_vivo, [6])
11 test_ds = test_ds.shuffle(buffer_size=BATCH_SIZE * 8, seed=SHUFFLE_SEED).batch(BATCH_SIZE)
12 test_ds = test_ds.map(lambda x, y: (add_noise(x, noises, scale=SCALE), y))
13 for audios, labels in test_ds.take(1):
14     # Obtener la señal FFT
15     fffts = audio_to_fft(audios)
16     # Predecir
17     y_pred = model.predict(ffffs)
18     max_value=0
19     idx_max_pred_value=0
20     name_max_pred_value=""
21     pred_name = class_names[np.argmax(y_pred)]
22     print(pred_name)
23     #print("2) -- {} () {}".format(index, y_pred) )
24     for index, value in enumerate(y_pred[0]):
25         #print("y_pred[0] <={}>".format(index,value) )
26         if(value>max_pred_value):
27             max_pred_value=value
28             idx_max_pred_value=index
29             name_max_pred_value=pred_name
30     print("1) proba.cercania={} orden_categoria={} nombre_categoria={}".format(max_pred_value, idx_max_pred_value, pred_name))
31     y_pred = np.argmax(y_pred, axis=-1)
32     print("2) predicción(Voz en Vivo) = {} ... y_pred={{} index={}}".format(class_names[y_pred[0]], y_pred, index))
33     print("3) registro (Voz en Vivo) = {} -{}".format(valid_audio_paths_vivo[0], y_pred, index))
34     displayAudio(valid_audio_paths_vivo[0], rate=SAMPLING_RATE)
35

```

0:01 / 0:01

CristinaFernandezDeKirchner
1) proba.cercania=[1.0] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav -

Ilustración 40: predicción para la prueba 1.

La predicción no fue acertada. El modelo devolvió con métrica accuracy: 100% (compatibilidad total) que la voz viva es la misma que la voz que se usó para entrenar al modelo.

Prueba 2 – entrenamiento con 1 hablante

El modelo es entrenado únicamente con la voz de un hablante.

La predicción se calcula a partir de una voz distinta a la única voz que entrenó el modelo. En este caso, la prueba es la misma que en el caso anterior, pero se utilizan menos archivos para entrenamiento.

El resultado esperado, es que el modelo vuelva a predecir con 100% de accuracy, es decir, compatibilidad absoluta entre la voz viva y la voz que se usó para entrenar el modelo.

Con VALID_SPLIT=0.9 (en lugar de 0.1):

```

23     print(pred_name)
24     #print("2) -- () {} ".format(pred_name, y_pred))
25     for index, value in enumerate(y_pred[0]):
26         #print("y_pred[0]{}={}<={})".format(index,value))
27         if(value>max_pred_value):
28             max_pred_value=value
29             idx_max_pred_value=index
30             name_max_pred_value=pred_name
31     print("1) proba.cercanias[{}].orden_categoria={} nombre_categoria={}{}".format(max_pred_value, idx_max_pred_value, name_max_pred_
32     y_pred = np.argmax(y_pred, axis=1)
33     print("2) prediccion[Voz en Vivo] = {} ... y_pred[0] index={})".format(class_names[y_pred[0]], y_pred, index))
34     print("3) registro [Voz en Vivo] = {} {}".format(valid_audio_paths_vivo[0], y_pred, ))
35     display(Audio(valid_audio_paths_vivo[0]), rate=SAMPLING_RATE)

```

CristinaFernandezDeKirchner
1) proba.cercanias[1].orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) prediccion[Voz en Vivo] = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro [Voz en Vivo] = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [132]

▶ 0:01/0:01 ⏪ ⏴

Ilustración 41: predicción para la prueba 2.

El resultado de la predicción, no cambió.

Prueba 3 – entrenamiento con 1 hablante.

El modelo es entrenado únicamente con la voz de un hablante.

La predicción se calcula a partir de una voz distinta a la única voz que entrenó el modelo. En este caso, se utilizan todavía menos archivos para entrenamiento que en el caso anterior (prueba 2). Con VALID_SPLIT=0.99 (en lugar de 0.1) - equivale a entrenar con 10 archivos.

Usando 10 archivos para entrenamiento:

```

In [24]: 1 history = model.fit(
2     train_ds,
3     epochs=EPOCHS,
4     validation_data=valid_ds,
5     callbacks=[earlystopping_cb, mdcheckpoint_cb],
6 )
10/10 [=====] - 12s 1s/step - loss: 0.5551 - accuracy: 0.9000 - val_loss: 1.6522e-08 - val_

```

```

In [24]: 1 print(model.evaluate(valid_ds))
1
30/30 [=====] - 9s 285ms/step - loss: 1.6522e-08 - accuracy: 1.0000
[1.652150061204383e-08, 1.0]

```

(GPU) del backend de Google Compute Engine que utiliza Python 3 Mostrando recursos desde las 16:09

Ilustración 42: métricas para la prueba 3.

El resultado tampoco cambió:

```

CristinaFernandezDeKirchner
1) proba.cercania=[1.0] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [132]

```

Ilustración 43: predicción para la prueba 3.

Prueba 4 – entrenamiento con 1 hablante.

El modelo es entrenado únicamente con la voz de un hablante.

La predicción se calcula a partir de una voz distinta a la única voz que entrenó el modelo. En este caso, se incrementa la cantidad de EPOCHS de 1 a 10, manteniendo VALID_SPLIT=0.99

```

1 #sobreescribo EPOCHS = 1
2 EPOCHS = 5
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdccheckpoint_cb],
9 )

```

Epoch 1/5
10/10 [=====] - 10s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 -
Epoch 2/5
10/10 [=====] - 9s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - v
Epoch 3/5
10/10 [=====] - 10s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 -
Epoch 4/5
10/10 [=====] - 11s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 -
Epoch 5/5
10/10 [=====] - 10s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 -

```

1 print(model.evaluate(valid_ds))

```

30/30 [=====] - 9s 282ms/step - loss: 0.0000e+00 - accuracy: 1.0000
[0.0, 1.0]

Ilustración 44: métricas para la prueba 4.

Se activó la condición de corte después de la quinta EPOCH.

Finalmente, el resultado no cambió.

```

CristinaFernandezDeKirchner
1) proba.cercania=[1.0] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [132]

```

Ilustración 45: predicción para la prueba 4.

A lo largo de las primeras 4 pruebas, se modificaron parámetros para determinar si las predicciones son compatibles con sobreentrenamiento.

En aprendizaje automático, el sobreajuste (también es frecuente emplear el término en inglés overfitting) es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado.

El algoritmo de aprendizaje debe alcanzar un estado en el que será capaz de predecir el resultado en otros casos a partir de lo aprendido con los datos de entrenamiento, generalizando para poder resolver situaciones distintas a las acaecidas durante el entrenamiento. Sin embargo, cuando un sistema se entrena demasiado (se sobreentrena) o se entrena con datos extraños, el algoritmo de aprendizaje puede quedar ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo.

Durante la fase de sobreajuste el éxito al responder las muestras de entrenamiento sigue incrementándose mientras que su actuación con muestras nuevas va empeorando.

En otras palabras, el modelo recuerda una gran cantidad de ejemplos en lugar de aprender a notar características.

¿Por qué devuelve 100% accuracy, si la voz viva es distinta a la voz que se usó para entrenar el modelo?

El método predict devuelve un array(s) Numpy de predicciones. Si el modelo entrenó con 1 sola categoría, entonces cualquier entrada va a estar probabilísticamente más cerca a dicha categoría.

Si el modelo hubiera entrenado con 2 categorías, entonces la probabilidad ya no va a ser absoluta. Sino que va a demostrar un nivel de certeza entre ambas distancias.

El caso es que hay que tener varias categorías para poder decidir a cuál se acerca más. Entonces, es un problema de distancias. Predict va a indicar con probabilidad, a qué categoría se acerca más.

Prueba 5 - imitador.

El modelo es entrenado con la voz de un hablante (voz actual y voz del pasado)

La Predicción se realiza con la voz “viva” del imitador.

- Notebook = voiceRecognition_v0010201
- Audios para entrenamiento = prueba_002.xls
- nombres_políticos = [CristinaFernandezDeKirchner, CristinaFernandezDeKirchner_hace22años]
- voz_viva = Fátima Florez

```
In [13]: 1 history = model.fit(  
2     train_ds,  
3     epochs=EPOCHS,  
4     validation_data=valid_ds,  
5     callbacks=[earlystopping_cb, mdlcheckpoint_cb],  
6 )  
Epoch 1/3  
10/10 [=====] - 281s 28s/step - loss: 1.5146 - accuracy: 0.6541 - val_loss: 0.4372 - val_accuracy: 0.8014  
Epoch 2/3  
10/10 [=====] - 230s 22s/step - loss: 0.2301 - accuracy: 0.9112 - val_loss: 0.0839 - val_accuracy: 0.9716  
Epoch 3/3  
10/10 [=====] - 218s 22s/step - loss: 0.0809 - accuracy: 0.9701 - val_loss: 0.0195 - val_accuracy: 0.9858  
  
In [14]: 1 print(model.evaluate(valid_ds))  
5/5 [=====] - 7s 1s/step - loss: 0.0195 - accuracy: 0.9858  
[0.019480828195810318, 0.9858155846595764]
```

Ilustración 46: métricas para la prueba 5.

Verificación:

```

1/1 [=====] - 0s 77ms/step
CristinaFernandezDeKirchner
1) proba.cercanía=[0.872580349445343] orden_categoria=109 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[109] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [109]

▶ 0:00 / 0:01 ━━━━ ◁ : 

```

Ilustración 47: predicción para la prueba 5.

La voz de Fátima Florez, tiene una cercanía del 87,258% con la de CFK.

Prueba 6 – audio atemporal (voz del pasado)

El modelo es entrenado con la voz de un hablante (voz actual y voz del pasado)

La Predicción se realiza con la voz “viva” del hablante en el pasado.

- Notebook = voiceRecognition_v0010201
- Audios para entrenamiento = prueba_002.xls
- nombres_políticos = [CristinaFernandezDeKirchner, CristinaFernandezDeKirchner_hace22años]
- voz_viva = CristinaFernandezDeKirchner_hace22años

```

1/1 [=====] - 0s 111ms/step
CristinaFernandezDeKirchner_hace22años
1) proba.cercanía=[0.9999834299087524] orden_categoria=58 nombre_categoria=CristinaFernandezDeKirchner_hace22años
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner_hace22años ... y_pred=[58] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [58]

▶ 0:00 / 0:01 ━━━━ ◁ : 

```

Ilustración 48: predicción para la prueba 6.

La voz de CFK de hace 22 años, tiene una cercanía de 99,999% a la de su propia

VOZ.

Prueba 7 – hablante de otro sexo sin entrenar el modelo

El modelo es entrenado con la voz de un hablante (voz actual y voz del pasado)

La Predicción se realiza con la voz “viva” del un hablante de otro sexo y que no entrenó el modelo.

- Notebook = voiceRecognition_v0010201
- Audios para entrenamiento = prueba_002.xls
- nombres_políticos = [CristinaFernandezDeKirchner, CristinaFernandezDeKirchner_hace22años]
- voz_viva = OscarParrilli



Ilustración 49: predicción para la prueba 7.

La voz de Oscar Parrilli, tiene una cercanía de 54,483% con la de CFK

Prueba 8 – Voz “similar” que no se usó para entrenar

El modelo es entrenado con la voz de un hablante (voz actual y voz del pasado)

La Predicción se realiza con la voz “viva” del un hablante que es aparentemente similar a la de la voz usada para entrenar el modelo.

- Notebook = voiceRecognition_v0010201
- Audios para entrenamiento = prueba_002.xls

- nombres_políticos = [CristinaFernandezDeKirchner, CristinaFernandezDeKirchner_hace22años]
- voz_viva = Victoria Tolosa Paz

```
1/1 [=====] - 0s 77ms/step
CristinaFernandezDeKirchner
1) proba.cercanía=[0.9976761937141418] orden_categoria=109 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[109] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [109]
```

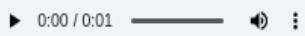


Ilustración 50: predicción para la prueba 8.

La voz de Victoria Tolosa Paz, tiene una cercanía de 99,767% con la de CFK.

Esta predicción es un error, como un caso de error tipo 1 (predijo con certeza una compatibilidad, pero en realidad no lo es)

¿Se debe entrenar el modelo con los registros de la voz viva?

La voz de “Victoria Tolosa Paz” no se usó durante el entrenamiento. Al predecir con la voz de un hablante que no se usó para entrenar al modelo, entonces la predicción puede fallar, siempre que se conserven las métricas que se mostraron en la prueba 5. El caso es que el modelo se entrenó relativamente con pocas muestras, sin diversidad de hablantes, en 3 EPOCHs.

Prueba 9 - Dos hablantes para entrenar el modelo

Se repite la prueba 8, pero se usa la voz que generó error tipo 1 para entrenar el modelo. Ahora, los dos hablantes se usan para entrenar el modelo.

La Predicción se realiza con las dos voces “vivas” de los hablantes que tienen una alta probabilidad de generar un error de tipo 1 entre sí.

Objetivo: En la última prueba 02, se detectaron 2 hablantes a muy corta distancia entre sí. En esta prueba, se usan esos 2 hablantes para entrenar el modelo, y 1 de las voces para poner a prueba la predicción.

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_003.xls
- nombres_políticos = [CristinaFernandezDeKirchner, VictoriaTolosaPaz]
- voz_viva = Victoria Tolosa Paz

```
In [66]: 1 history = model.fit(
2     train_ds,
3     epochs=EPOCHS,
4     validation_data=valid_ds,
5     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
6 )
Epoch 1/3
1281/1281 [=====] - 269s 207ms/step - loss: 0.3379 - accuracy: 0.9141 - val_loss: 0.0113
- val_accuracy: 1.0000
Epoch 2/3
1281/1281 [=====] - 265s 206ms/step - loss: 0.4256 - accuracy: 0.9586 - val_loss: 0.0741
- val_accuracy: 0.9789
Epoch 3/3
1281/1281 [=====] - 265s 207ms/step - loss: 0.1001 - accuracy: 0.9641 - val_loss: 0.0846
- val_accuracy: 0.9859

In [67]: 1 print(model.evaluate(valid_ds))
5/5 [=====] - 8s 1s/step - loss: 0.0846 - accuracy: 0.9859
[0.08461076021194458, 0.98591548204422]
```

Ilustración 51: métricas para la prueba 9.

Verificación:

```
1/1 [=====] - 1s 564ms/step
VictoriaTolosaPaz
1) proba.cercanía=[0.9363936185836792] orden_categoria=77 nombre_categoria=VictoriaTolosaPaz
2) predicción(Voz en Vivo) = VictoriaTolosaPaz ... y_pred=[77] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [77]
```



Ilustración 52: predicción para la prueba 9.

La voz de Victoria Tolosa Paz, tiene una cercanía de 93,639% con la de ella misma .

Prueba 10 - Dos hablantes para entrenar el modelo

Se repite la prueba 9. La Predicción se realiza con la otra voz “viva” que tiene una alta probabilidad de generar un error de tipo 1 entre respecto a la otra voz.

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_003.xls
- nombres_políticos = [CristinaFernandezDeKirchner, VictoriaTolosaPaz]
- voz_viva = CristinaFernandezDeKirchner

```
1/1 [=====] - 0s 91ms/step
CristinaFernandezDeKirchner
1) proba.cercanía=[0.9999275207519531] orden_categoria=109 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernándezDeKirchner ... y_pred=[109] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [109]
```

Ilustración 53: predicción para la prueba 10.

La voz de CFK tiene una cercanía de 99,992% con la de ella misma.

Prueba 11 - Dos hablantes para entrenar el modelo

Se repite la prueba 9. La Predicción se realiza con voz “viva” del imitador que tiene una alta probabilidad de generar un error de tipo 1 entre respecto a las otras voces.

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_003.xls
- nombres_políticos = [CristinaFernandezDeKirchner, VictoriaTolosaPaz]
- voz_viva = FatimaFlorez

```

1/1 [=====] - 0s 77ms/step
CristinaFernandezDeKirchner
1) proba.cercanía=[0.872580349445343] orden_categoria=109 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[109] index=132
3) registro (Voz en Vivo) = /home/jmmiguez/proyectoAudio/audio//audio/VozViva/VozViva.wav - [109]

▶ 0:00 / 0:01 ━━━━ ⏪ ⏴ :
```

Ilustración 54: predicción para la prueba 11.

La voz viva de Fátima Florez tiene una cercanía de 87,258% con la de CFK

Prueba 12 - Pruebas de rendimiento - Localhost

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_004.xls
- nombres_políticos = {todos}
- voz_viva = no aplica.

Procesando desde el 29/08/2022 23:26, con 3 EPOCHS.

```

In [73]: 1 history = model.fit(
2     train_ds,
3     epochs=EPOCHS,
4     validation_data=valid_ds,
5     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
6 )

Epoch 1/3
47115/47115 [=====] - 10702s 227ms/step - loss: 4.6586 - accuracy: 0.0554 - val_loss: 4.6
529 - val_accuracy: 0.0604
Epoch 2/3
47115/47115 [=====] - 10013s 213ms/step - loss: 4.6477 - accuracy: 0.0557 - val_loss: 4.6
514 - val_accuracy: 0.0604
Epoch 3/3
47115/47115 [=====] - 9309s 198ms/step - loss: 4.6464 - accuracy: 0.0557 - val_loss: 4.65
11 - val_accuracy: 0.0604

In [74]: 1 print(model.evaluate(valid_ds))

164/164 [=====] - 268s 2s/step - loss: 4.6511 - accuracy: 0.0604
[4.651064872741699, 0.06036294251680374]
```

Ilustración 55: métricas para la prueba de rendimiento Localhost

Con 3 EPOCHS, Accuracy es menor al 10%. El resultado es inaceptable.

¿Cuántas EPOCHS necesito para procesar esta cantidad de registros?

Se analizará si se pueden sugerir 100 EPOCHS y un corte por estabilidad.

La prueba 12 muestra que **no hay que apurar el procesamiento** para obtener la evidencia.

El tiempo de cada EPOCH es 10702 segundos, equivalente a 2.97 horas / EPOCH. Esto quiere decir, que con este equipo de procesamiento, 3 EPOCHS demoran 30024 segundos, equivalente a 8.34 horas. Este tiempo no es aceptable.

Equipo de procesamiento local

Procesador: AMD® A4-4000 apu with radeon(tm) hd graphics × 2

Capacidad de disco: 980,2 GB

Memoria: 10GiB

Nombre y Tipo de SO: Ubuntu 22.04 LTS 64 bits

Gráficos: AMD® Aruba

Software

python3

pip + pip3

Anaconda Navigator 2.2.0 con Jupyter Notebook

Tabla 7: Características de HW y SW para pruebas de rendimiento Localhost

Se puede apreciar que al momento de hacer las validaciones, equivoca todas las predicciones (las métricas no fueron buenas)

```
4/4 [=====] - 8s 2s/step
rnd [ 71 90 55 35 23 98 124 80 96 127]
Speaker: JimenaLatorre Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: BeatrizAvila Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: JulianaDiTullio Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: OmarDeMarchi Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: AlvaroMartinez Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: JuanManuelLopez Predicted: JoseMayans
▶ 0:00 / 0:01 ━━━━ ◉ : 
Speaker: VictorZimmermann Predicted: JoseMayans
```

Ilustración 56: resultado para pruebas de rendimiento Localhost

Prueba 13 – Rendimiento, GPU, 1 Epochs, split 0,1

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = "/home/jmmiguez/proyectoAudio/audio/prueba_004.xls"
- nombres_políticos = {todos}
- voz_viva = x

Procesando con 1 EPOCH en Google Colab, con GPU:

```
VALID_SPLIT = 0.1
```

```
EPOCHS = 1
```

La velocidad de procesamiento con GPU, para 1 EPOCH, demora 5:36horas.

Se estima que el procesamiento completo va a demorar más de 12 horas, por lo que se interrumpe el procesamiento.

Prueba 14 - Rendimiento, TPU, Epochs 1, split 0,1

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_004.xls
- nombres_políticos = {todos}
- voz_viva = x

Procesando con 1 EPOCH en Google Colab, con TPU:

```
VALID_SPLIT = 0.1
```

```
EPOCHS = 1
```

Verificando si estoy procesando con TPU:

```

+ Código + Texto
[9] Total params: 3,105,238
Trainable params: 3,105,238
Non-trainable params: 0

1 #sobreescribe EPOCHS = 1
2 EPOCHS = 1
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
9 )

```

Sesiones activas

Título	Última ejecución	RAM usada	Opciones
procesamiento_V0020101.ipynb Sesión actual	TPU hace 1 minuto	1.50 GB	FINALIZAR
procesamiento_V0020101.ipynb	hace 5 minutos	1.50 GB	FINALIZAR

FINALIZAR OTRAS SESIONES CERRAR

Ilustración 57: Verificando procesamiento en colab con TPU

```

+ Código + Texto
[9] Total params: 3,105,238
Trainable params: 3,105,238
Non-trainable params: 0

1 #sobreescribe EPOCHS = 1
2 EPOCHS = 1
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
9 )

```

3/351 [.....] - ETA: 1:11:33 - loss: 5.0358 - accuracy: 0.0391

Ilustración 58: métricas pruebas de rendimiento colab, TPU, 1 EPOCH, split 0,1

El procesamiento de 1 EPOCH demora 1:11horas.

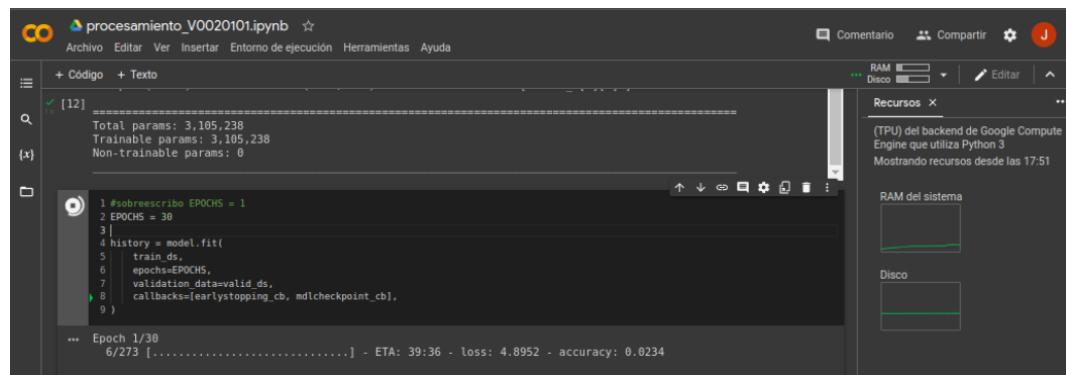
Prueba 15 - Rendimiento, TPU, 30 Epochs, split 0,3

- Notebook = voiceRecognition_v0010201
- xls_entrenamiento = prueba_004.xls
- nombres_políticos = {todos}
- voz_viva = x

VALID_SPLIT = 0.3

Usando 34886 archivos para entrenamiento.

Usando 14950 archivos para validación.



The screenshot shows a Jupyter Notebook cell with the code:

```
1 #obreescribo EPOCHS = 1
2 EPOCHS = 30
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
9 )
```

Output from the cell:

```
Total params: 3,105,238
Trainable params: 3,105,238
Non-trainable params: 0
```

... Epoch 1/30
6/273 [.....] - ETA: 39:36 - loss: 4.8952 - accuracy: 0.0234

On the right, there is a "Recursos" panel showing system RAM and disk usage.

Ilustración 59: métricas rendimiento colab, TPU, 30 EPOCHs, split 0,3

Cada EPOCH tiene una duración estimada de 0:39horas. Por esto, se finaliza intencionalmente el procesamiento (ya que el procesamiento completo supera las 12 horas)

Prueba 16 - Rendimiento, TPU, 10 Epochs, split 0,4

Configuración:

- VALID_SPLIT = 0.4
- Usando 29902 archivos para entrenamiento.
- Usando 19934 archivos para validación.
- EPOCHS = 10

El procesamiento de 1 EPOCH demora 33 minutos

```

1 #sobreescribe EPOCHS = 1
2 EPOCHS = 10
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdccheckpoint_cb],
9 )

```

... Epoch 1/10
9/234 [>.....] - ETA: 33:12 - loss: 5.0782 - accuracy: 0.0087

Ilustración 60: métricas rendimiento colab, TPU, 1 EPOCH, split 0,4

Tiene que demorar menos de 12 horas, tiempo en que la conexión se pierde por políticas de uso de Google: <https://research.google.com/colaboratory/faq.html#gpu-availability> y <https://research.google.com/colaboratory/faq.html#tpu-availability>

¿Qué tipos de GPU están disponibles en Colab?

Los tipos de GPU disponibles en Colab varían con el tiempo. Esto es necesario para que Colab pueda brindar acceso a estos recursos sin costo. Si necesitas un acceso más estable a las GPU más veloces de Colab, puedes probar Colab Pro y Pro+. Si deseas utilizar hardware específico en Colab, consulta las VM de GCP Marketplace de Colab.

Ten en cuenta que no se permite el uso de Colab para la minería de criptomonedas, y esto podría causar que se te prohíba usar Colab por completo.

¿Durante cuánto tiempo se pueden ejecutar los notebooks en Colab?

Para ejecutarse, los notebooks se conectan a máquinas virtuales que tienen una vida útil máxima de hasta 12 horas. Además, se desconectan los notebooks de las VM cuando permanecen inactivos durante un período prolongado. La vida útil máxima de las VM y el tiempo de espera de inactividad pueden variar con el tiempo o en función de tu uso. Esto es necesario para que Colab pueda ofrecer recursos computacionales sin costo. Los usuarios que buscan VM con una vida útil más prolongada y tiempos de espera de inactividad más flexibles que no varíen demasiado con el tiempo, pueden probar Colab Pro y Pro+.

Si deseas administrar la vida útil de tu VM de Colab, consulta las VM de GCP Marketplace de Colab que te proporcionan un entorno persistente y adaptable a tus preferencias.

En la documentación de Google, cambiando el literal “TPU” por “GPU” permite observar que el tiempo de ejecución permitido es el mismo en ambos casos.

Entonces, tengo que conseguir que el procesamiento no supere 12horas.

A 30 minutos por EPOCH, y 10 EPOCHS, son 300 minutos, que equivalen a 50 horas.

Se interrumpe nuevamente y se vuelve a configurar.

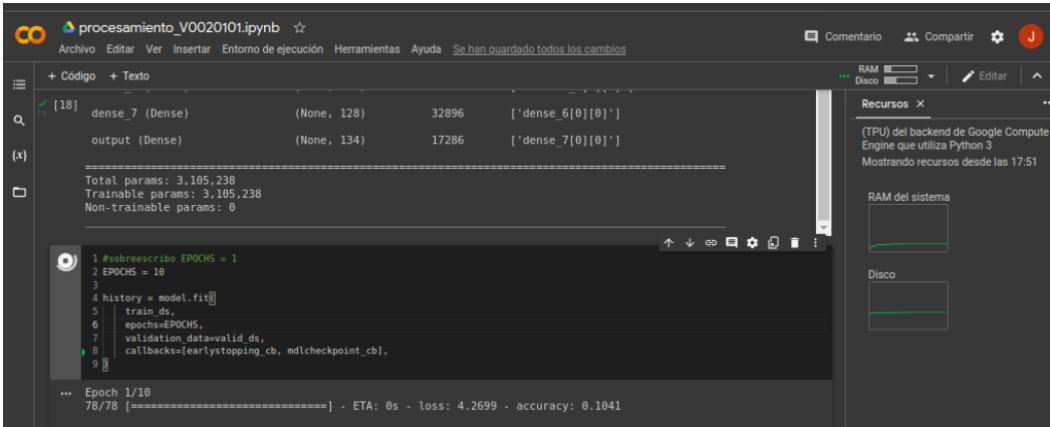
Prueba 17 - Rendimiento, TPU, 10 Epochs, split 0,8

- VALID_SPLIT = 0.8
- EPOCHS = 10

El procesamiento de 1 EPOCH demora hasta 11 minutos.

Usando 9968 archivos para entrenamiento.

Usando 39868 archivos para validación.



```
[18]: dense_7 (Dense)      (None, 128)      32896      ['dense_6[0][0]']
         output (Dense)     (None, 134)      17286      ['dense_7[0][0]']

=====
Total params: 3,105,238
Trainable params: 3,105,238
Non-trainable params: 0

1 # sobreescribe EPOCHS = 10
2 EPOCHS = 10
3
4 history = model.fit()
5   train.ds
6   epochs=EPOCHS,
7   validation_data=valid.ds,
8   callbacks=[earlystopping_cb, mdccheckpoint_cb],
9

*** Epoch 1/10
78/78 [=====] - ETA: 0s - loss: 4.2699 - accuracy: 0.1041
```

Ilustración 61: métricas pruebas de rendimiento colab, TPU, 10 EPOCH, split 0,8

La prueba demora 11 minutos x cada EPOCH. Equivale a 110 minutos. Esto equivale a 1,8 horas.

Después de varias horas, el procesamiento no avanzó. Se reinicia, pero finalizado primer EPOCH, el procesamiento no avanza. A efectos prácticos, se procedió a terminar el procesamiento de la prueba 17.

Para las siguientes pruebas, se selecciona menor cantidad de muestras de hablantes para poder continuar hacia el logro del objetivo.

Prueba 18 – Rendimiento: voces género Femenino

En este caso, el archivo prueba_005.xls contiene los nombres de hablantes de género femenino. El archivo consta de 45 hablantes.

39	7191	7898	MariaTeresaGonzalez
40	8150	8321	AlejandraVigo
41	10359	10808	LuciaCorpacci
42	14799	15393	BeatrizAvila
43	21657	21699	MariaEugeniaCatalfamo
44	21912	22208	MariaEugeniaDure
45	22256	22774	MariaClaraVega
46	3900	5100	CristinaFernandezDeKirchner

Tabla 8: Tabla de voces de hablantes de género femenino para Rendimiento.

- VALID_SPLIT = 0.1
- EPOCHS = 5

```
4 history = model.fit(  
5     train_ds,  
6     epochs=EPOCHS,  
7     validation_data=valid_ds,  
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],  
9 )  
  
Epoch 1/5  
1/105 [.....] - ETA: 3:22:07 - loss: 5.0350 - accuracy: 0.0391
```

Ilustración 62: métrica inicial, rendimiento colab, TPU, 5 EPOCHs, split 0,1

Inicialmente, estimó 1 EPOCH = 3:22horas. Haciendo cálculos, se estima que 5 EPOCHS $\sim 3.3 \times 5 > 12$ horas (al ser mayor que 12 horas, finaliza el procesamiento)

```
4 history = model.fit(  
5     train_ds,  
6     epochs=EPOCHS,  
7     validation_data=valid_ds,  
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],  
9 )  
  
Epoch 1/5  
15/105 [==>.....] - ETA: 18:50 - loss: 4.4485 - accuracy: 0.0651
```

Ilustración 63: métrica, rendimiento colab, TPU, 5 EPOCHs, split 0,1

5 minutos después, estimó 1 EPOCH = 18:50 minutos; 5 EPOCHS $\sim 1,57$ horas.

Esta prueba, se usa como patrón para fijar la cantidad máxima de hablantes que se usará para entrenar el modelo.

Se evalúa si con 5 EPOCHS, la métrica accuracy es aceptable.

Mientras tanto, performance con TPU consumidos:

Memoria: 1.61GB/12GB

Disco: 37.48GB/107.72GB

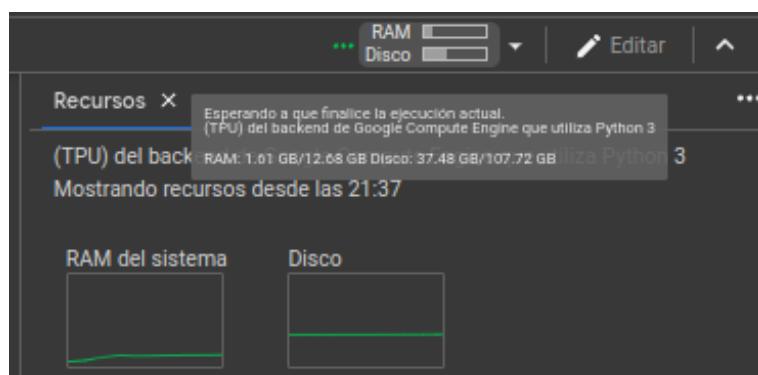


Ilustración 64: Prueba 18 - recursos consumidos.

A screenshot of a terminal window displaying training metrics. The code at the top shows a call to 'model.fit' with parameters: 'train_ds', 'epochs=EPOCHS', 'validation_data=valid_ds', and 'callbacks=[earlystopping_cb, mdlicheckpoint_cb]'. Below the code, the output shows the training progress for 105 epochs. The output includes the epoch number, the number of steps per epoch, and various metrics like loss, accuracy, and validation loss and accuracy. The metrics show an upward trend from epoch 1 to epoch 105.

Ilustración 65: prueba 18, Métricas por Epochs.

- en 1 EPOCH se alcanzó ~ 0.48 accuracy
- en 2 EPOCHS se alcanzó ~ 0.54 accuracy
- en 3 EPOCHS se alcanzó ~ 0.56 accuracy

- en 4 EPOCHS se alcanzó ~ 0.64 accuracy - 19 minutos promedio por EPOCH
- en 5 EPOCHS se alcanzó ~ 0.65 accuracy

```

41 s
1 print(model.evaluate(valid_ds))
47/47 [=====] - 395 793ms/step - loss: 1.1698 - accuracy: 0.6528
[1.1698479652404785, 0.6527590751647949]

```

Ilustración 66: Métricas para la prueba 18.

Speaker	Predicted
AnaCarolinaGaillard	AnaCarolinaGaillard
CristinaFernandezDeKirchner	CristinaFernandezDeKirchner
SilvanaGinocchio	SilvanaGinocchio
SilvinaGarciaLarraburu	SilvinaGarciaLarraburu
MargaritaStolbizer	MariaVictoriaTejeda
AnahiCosta	GabrielaLena
MyriamBregman	MyriamBregman
MariaVictoriaTejeda	MariaVictoriaTejeda
NataliaZaracho	NataliaZaracho
AnahiCosta	MyriamBregman

Ilustración 67: Resultados para la prueba 18.

Probando con la VozViva de CFK: con accuracy = 99,99%

```
▷ CristinaFernandezDeKirchner
1) proba.cercanía=[0.9999990463256836] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [132]

▶ 0:00 / 0:01 ━━━━ ⏪ ⏴
```

Ilustración 68: predicción para la prueba 18.

Prueba 19 – Rendimiento: voces género Femenino

Esta prueba es similar a la Prueba 18, pero se utiliza la voz viva de un hablante con similitud a otro hablante.

- voz_viva = VictoriaTolosaPaz_igual_a_CFK

```
▷ VictoriaTolosaPaz
1) proba.cercanía=[0.9989763498306274] orden_categoria=92 nombre_categoria=VictoriaTolosaPaz
2) predicción(Voz en Vivo) = VictoriaTolosaPaz ... y_pred=[92] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [92]

▶ 0:00 / 0:01 ━━━━ ⏪ ⏴
```

Ilustración 69: predicción para la prueba 19.

Prueba 20 – Rendimiento: voces género Femenino

Aunque el modelo se entrenó con voces Femeninas, se prueba con voces de género Masculino.

- voz_viva = JavierMilei

```
▷ BlancaOsuna
1) proba.cercanía=[0.2331719994544983] orden_categoria=52 nombre_categoria=BlancaOsuna
2) predicción(Voz en Vivo) = BlancaOsuna ... y_pred=[52] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [52]

▶ 0:00 / 0:01 ━━━━ ⏪ ⏴
```

Ilustración 70: predicción para la prueba 20.

Prueba 21 – Rendimiento: voces género Femenino

Aunque el modelo se entrenó con voces Femeninas, Se prueba con voces de género Masculino.

- voz_viva = Juan Manuel Miguez (_jmm_blablablabla) / es un audio de un masculino, con la frase literal: “blablablabla”

```
↳ LuciaCorpacci
1) proba.cercanía=[0.9185916185379028] orden_categoria=100 nombre_categoria=LuciaCorpacci
2) predicción(Voz en Vivo) = LuciaCorpacci ... y_pred=[100] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [100]
```

The screenshot shows a Jupyter Notebook cell with the code above. Below the code is a media player interface with a play button, a progress bar at 0:01/0:01, and a volume icon. At the bottom right of the cell are two buttons: '+ Código' and '+ Texto'.

Ilustración 71: predicción para la prueba 21.

Prueba 22 – Rendimiento: voces género Femenino

Aunque el modelo se entrenó con voces Femeninas, Se prueba con una voz atemporal (voz viva del pasado)

- voz_viva = CFK_hace20años:

```
↳ MyriamBregman
1) proba.cercanía=[0.694869339466095] orden_categoria=20 nombre_categoria=MyriamBregman
2) predicción(Voz en Vivo) = MyriamBregman ... y_pred=[20] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [20]
```

The screenshot shows a Jupyter Notebook cell with the code above. Below the code is a media player interface with a play button, a progress bar at 0:00/0:01, and a volume icon. At the bottom right of the cell are two buttons: '+ Código' and '+ Texto'.

Ilustración 72: predicción para la prueba 22.

Prueba 23 – Rendimiento: voces género Femenino

El modelo se entrenó con voces Femeninas y se prueba con la voz de un imitador.

- voz_viva = FatimaFlorez

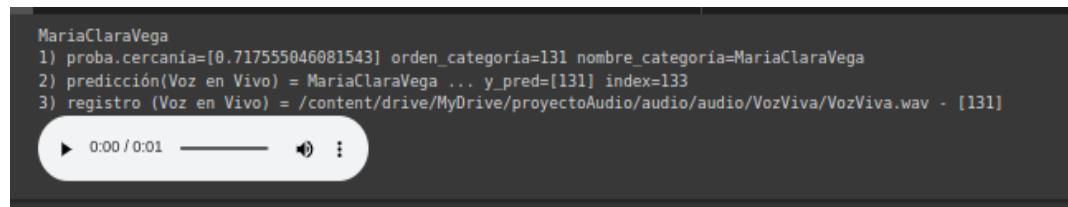


Ilustración 73: predicción para la prueba 23.

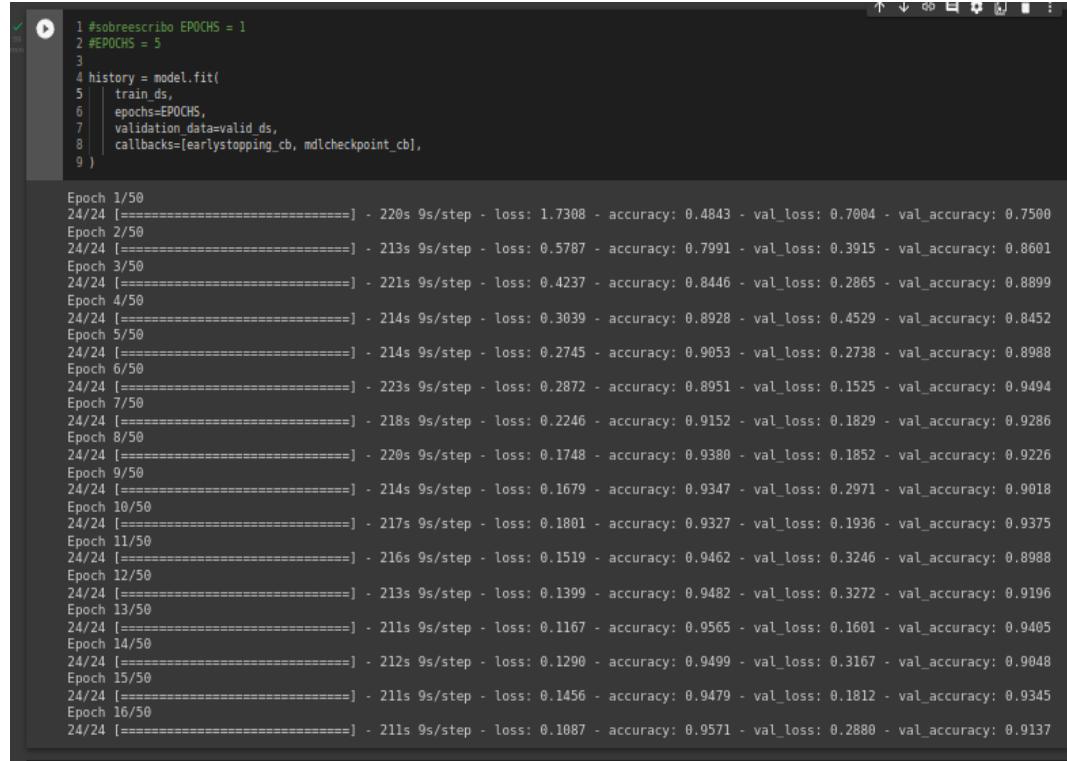
Prueba 24 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = CristinaFernandezDeKirchner

	A	B	C	D
1	segs_ini	segs_fin	Nombre_Diputado	
2	3242	3882	MargaritaStolbizer	
3	4525	5355	GracielaCamaño	
4	6123	7155	MyriamBregman	
5	35514	36010	VictoriaTolosaPaz	
6	3900	5100	CristinaFernandezDeKirchner	
7				

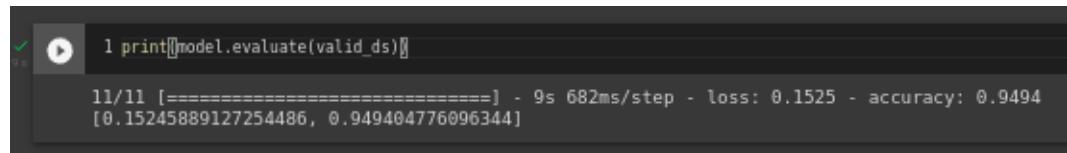
Tabla 9: Tabla de voces de hablantes de género femenino (set reducido de 5 hablantes)



```
1 #sobreescribo EPOCHS = 1
2 #EPOCHS = 5
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
9 )
```

Epoch 1/50
24/24 [=====] - 220s 9s/step - loss: 1.7308 - accuracy: 0.4843 - val_loss: 0.7004 - val_accuracy: 0.7500
Epoch 2/50
24/24 [=====] - 213s 9s/step - loss: 0.5787 - accuracy: 0.7991 - val_loss: 0.3915 - val_accuracy: 0.8601
Epoch 3/50
24/24 [=====] - 221s 9s/step - loss: 0.4237 - accuracy: 0.8446 - val_loss: 0.2865 - val_accuracy: 0.8899
Epoch 4/50
24/24 [=====] - 214s 9s/step - loss: 0.3039 - accuracy: 0.8928 - val_loss: 0.4529 - val_accuracy: 0.8452
Epoch 5/50
24/24 [=====] - 214s 9s/step - loss: 0.2745 - accuracy: 0.9053 - val_loss: 0.2738 - val_accuracy: 0.8988
Epoch 6/50
24/24 [=====] - 223s 9s/step - loss: 0.2872 - accuracy: 0.8951 - val_loss: 0.1525 - val_accuracy: 0.9494
Epoch 7/50
24/24 [=====] - 218s 9s/step - loss: 0.2246 - accuracy: 0.9152 - val_loss: 0.1829 - val_accuracy: 0.9286
Epoch 8/50
24/24 [=====] - 220s 9s/step - loss: 0.1748 - accuracy: 0.9380 - val_loss: 0.1852 - val_accuracy: 0.9226
Epoch 9/50
24/24 [=====] - 214s 9s/step - loss: 0.1679 - accuracy: 0.9347 - val_loss: 0.2971 - val_accuracy: 0.9018
Epoch 10/50
24/24 [=====] - 217s 9s/step - loss: 0.1801 - accuracy: 0.9327 - val_loss: 0.1936 - val_accuracy: 0.9375
Epoch 11/50
24/24 [=====] - 216s 9s/step - loss: 0.1519 - accuracy: 0.9462 - val_loss: 0.3246 - val_accuracy: 0.8988
Epoch 12/50
24/24 [=====] - 213s 9s/step - loss: 0.1399 - accuracy: 0.9482 - val_loss: 0.3272 - val_accuracy: 0.9196
Epoch 13/50
24/24 [=====] - 211s 9s/step - loss: 0.1167 - accuracy: 0.9565 - val_loss: 0.1601 - val_accuracy: 0.9405
Epoch 14/50
24/24 [=====] - 212s 9s/step - loss: 0.1290 - accuracy: 0.9499 - val_loss: 0.3167 - val_accuracy: 0.9048
Epoch 15/50
24/24 [=====] - 211s 9s/step - loss: 0.1456 - accuracy: 0.9479 - val_loss: 0.1812 - val_accuracy: 0.9345
Epoch 16/50
24/24 [=====] - 211s 9s/step - loss: 0.1087 - accuracy: 0.9571 - val_loss: 0.2880 - val_accuracy: 0.9137

Ilustración 74: Entrenamiento para la prueba 24.



```
1 print(model.evaluate(valid_ds))
```

11/11 [=====] - 9s 682ms/step - loss: 0.1525 - accuracy: 0.9494
[0.15245889127254486, 0.949404776096344]

Ilustración 75: Métricas para la prueba 24.

Accuracy ~ 0.95

```

rnd [ 92 90 107 22 66 39 77 11 117 87]
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ⏪ ⏴

```

Ilustración 76: Resultados para la prueba 24.

```

CristinaFernandezDeKirchner
1) proba.cercanía=[1.0] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner
2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [132]
▶ 0:00/0:01 ━━━━ ⏪ ⏴

```

Ilustración 77: predicción para la prueba 24.

Prueba 25 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = VictoriaTolosaPaz_igual_a_CFK

```

C: VictoriaTolosaPaz
1) proba.cercania=[0.9998331069946289] orden_categoria=92 nombre_categoria=VictoriaTolosaPaz
2) predicción(Voz en Vivo) = VictoriaTolosaPaz ... y_pred=[92] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [92]

```

Ilustración 78: predicción para la prueba 25.

Prueba 26 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = Javier Milei

```

C: MargaritaStolbizer
1) proba.cercania=[0.7302886843681335] orden_categoria=16 nombre_categoria=MargaritaStolbizer
2) predicción(Voz en Vivo) = MargaritaStolbizer ... y_pred=[16] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [16]

```

Ilustración 79: predicción para la prueba 26.

Prueba 27 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = JuanManuelMiguez (_jmm_blablablabla)

```

MyriamBregman
1) proba.cercania=[0.4980545938014984] orden_categoria=20 nombre_categoria=MyriamBregman
2) predicción(Voz en Vivo) = MyriamBregman ... y_pred=[20] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [20]

```

Ilustración 80: predicción para la prueba 27.

Prueba 28 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = _CFK_hace20años: (*predicción errónea*)

The screenshot shows a dark-themed digital interface. At the top, there is a status bar with the text: "MargaritaStolbizer", "1) proba.cercanía=[0.8173344731330872] orden_categoria=16 nombre_categoria=MargaritaStolbizer", "2) predicción(Voz en Vivo) = MargaritaStolbizer ... y_pred=[16] index=133", and "3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [16]". Below this is a media player control bar with a play button, a progress bar showing "0:00 / 0:01", and other standard controls.

Ilustración 81: predicción para la prueba 28.

Prueba 29 – voces género Femenino

El modelo se entrena con voces género Fememino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = _FatimaFlorez

The screenshot shows a dark-themed digital interface. At the top, there is a status bar with the text: "CristinaFernandezDeKirchner", "1) proba.cercanía=[0.5614812970161438] orden_categoria=132 nombre_categoria=CristinaFernandezDeKirchner", "2) predicción(Voz en Vivo) = CristinaFernandezDeKirchner ... y_pred=[132] index=133", and "3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [132]". Below this is a media player control bar with a play button, a progress bar showing "0:00 / 0:01", and other standard controls.

Ilustración 82: predicción para la prueba 29.

Prueba 30 – voces género Femenino

El modelo se entrena con voces género Femenino, con un set reducido y más EPOCHS que en las pruebas 18 a 23.

- EPOCHS = 50
- VALID_SPLIT = 0.1
- VozViva = _MariaEugeniaVidal:

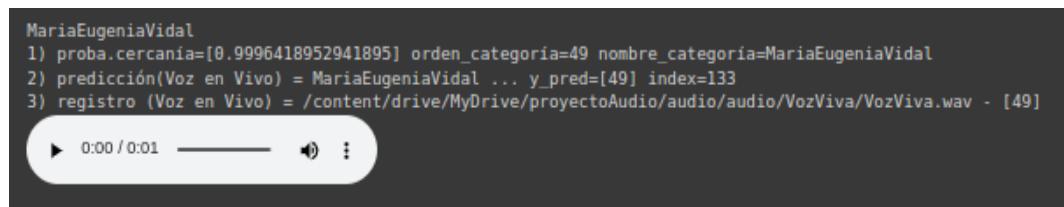


Ilustración 83: predicción para la prueba 30.

- ✓ La mayoría de las voces que se usaron en el entrenamiento, fueron reconocidas casi con certeza.
- ✓ Las voces que no se usaron para entrenar, no fueron reconocidas.

Resumen de la pruebas 24 a 30

- TPU
- VALID_SPLIT = 0.1
- EPOCHS = 50
- Modelo estable en 16 EPOCHS, en 3448 segs (57.5 minutos)
- loss: 0.1525 - accuracy: 0.9494

Prueba 31 – voces género Femenino, split 0.5

- procesamiento_V0020102.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.5
- EPOCHS = 50

```
+ Código + Texto
[15] Epoch 14/50
[14] Epoch 15/50
[14] Epoch 16/50
[14] Epoch 17/50
[14] Epoch 18/50
[14] Epoch 19/50
[14] Epoch 20/50
[14] Epoch 21/50
[14] Epoch 22/50
[14] Epoch 23/50
[14] Epoch 24/50
[14] Epoch 25/50
[14] Epoch 26/50
[14] Epoch 27/50
[14] Epoch 28/50
[14] Epoch 29/50
[14] Epoch 30/50
[14] Epoch 31/50
[14] Epoch 32/50
[14] Epoch 33/50
[14] Epoch 34/50
[14] Epoch 35/50
[14] Epoch 36/50
[14] Epoch 37/50
  14/50 [=====] - 154s 11s/step - loss: 0.222 - accuracy: 0.9240 - val_loss: 0.2052 - val_accuracy: 0.8990
  14/14 [=====] - 154s 11s/step - loss: 0.1660 - accuracy: 0.9412 - val_loss: 0.3092 - val_accuracy: 0.8925
  Epoch 15/50 [=====] - 156s 11s/step - loss: 0.1281 - accuracy: 0.9495 - val_loss: 0.3124 - val_accuracy: 0.9020
  Epoch 16/50 [=====] - 156s 11s/step - loss: 0.1126 - accuracy: 0.9626 - val_loss: 0.3369 - val_accuracy: 0.8978
  Epoch 17/50 [=====] - 156s 11s/step - loss: 0.1232 - accuracy: 0.9555 - val_loss: 0.5641 - val_accuracy: 0.8414
  Epoch 18/50 [=====] - 159s 11s/step - loss: 0.1094 - accuracy: 0.9626 - val_loss: 0.2387 - val_accuracy: 0.9222
  Epoch 19/50 [=====] - 156s 11s/step - loss: 0.1181 - accuracy: 0.9507 - val_loss: 0.4291 - val_accuracy: 0.8877
  Epoch 20/50 [=====] - 157s 11s/step - loss: 0.1260 - accuracy: 0.9513 - val_loss: 0.1807 - val_accuracy: 0.9424
  Epoch 21/50 [=====] - 154s 11s/step - loss: 0.1808 - accuracy: 0.9371 - val_loss: 0.3320 - val_accuracy: 0.8913
  Epoch 22/50 [=====] - 156s 11s/step - loss: 0.1087 - accuracy: 0.9572 - val_loss: 0.3561 - val_accuracy: 0.8972
  Epoch 23/50 [=====] - 155s 11s/step - loss: 0.1696 - accuracy: 0.9388 - val_loss: 0.1799 - val_accuracy: 0.9293
  Epoch 24/50 [=====] - 160s 11s/step - loss: 0.0932 - accuracy: 0.9691 - val_loss: 0.1819 - val_accuracy: 0.9430
  Epoch 25/50 [=====] - 157s 11s/step - loss: 0.1030 - accuracy: 0.9632 - val_loss: 0.3560 - val_accuracy: 0.8907
  Epoch 26/50 [=====] - 156s 11s/step - loss: 0.1022 - accuracy: 0.9584 - val_loss: 0.2478 - val_accuracy: 0.9340
  Epoch 27/50 [=====] - 157s 11s/step - loss: 0.1381 - accuracy: 0.9537 - val_loss: 0.1667 - val_accuracy: 0.9483
  Epoch 28/50 [=====] - 156s 11s/step - loss: 0.1349 - accuracy: 0.9519 - val_loss: 0.4010 - val_accuracy: 0.8705
  Epoch 29/50 [=====] - 158s 11s/step - loss: 0.1096 - accuracy: 0.9650 - val_loss: 0.2363 - val_accuracy: 0.9400
  Epoch 30/50 [=====] - 157s 11s/step - loss: 0.0999 - accuracy: 0.9656 - val_loss: 0.2340 - val_accuracy: 0.9293
  Epoch 31/50 [=====] - 155s 11s/step - loss: 0.0827 - accuracy: 0.9691 - val_loss: 0.2633 - val_accuracy: 0.9263
  Epoch 32/50 [=====] - 154s 11s/step - loss: 0.0380 - accuracy: 0.9887 - val_loss: 0.3739 - val_accuracy: 0.9228
  Epoch 33/50 [=====] - 155s 11s/step - loss: 0.0848 - accuracy: 0.9733 - val_loss: 0.3760 - val_accuracy: 0.9085
  Epoch 34/50 [=====] - 158s 11s/step - loss: 0.0738 - accuracy: 0.9762 - val_loss: 0.2577 - val_accuracy: 0.9204
  Epoch 35/50 [=====] - 157s 11s/step - loss: 0.0510 - accuracy: 0.9822 - val_loss: 0.2746 - val_accuracy: 0.9317
  Epoch 36/50 [=====] - 155s 11s/step - loss: 0.0389 - accuracy: 0.9875 - val_loss: 0.4294 - val_accuracy: 0.9204
  Epoch 37/50 [=====] - completed a las 16:35
```

Ilustración 84: Entrenamiento para la prueba 31.

Resumen de la prueba

- VALID_SPLIT = 0.5
- EPOCHS = 50
- Modelo estable en 37 EPOCHS, 4532 segs (equivalente a 75.53 minutos)

- loss: 0.1667 - accuracy: 0.9483
- TPU

```

+ Código + Texto
Speaker: MyriamBregman Predicted: MargaritaStolbizer
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
(x) Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: MariaEugenioVidal Predicted: MariaEugenioVidal
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: VictoriaTelosaPaz Predicted: VictoriaTelosaPaz
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴
Speaker: VictoriaTelosaPaz Predicted: VictoriaTelosaPaz
▶ 0:00 / 0:01 ━━━━ ⏷ ⏴

```

Ilustración 85: Resultados para la prueba 31.

Prueba 32 – voces género Femenino, split 0.2

- procesamiento_V0020103.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.2
- EPOCHS = 50

```

+ Código + Texto
  6 epochs=EPOCHS,
  7 validation_data=valid_ds,
  8 callbacks=[earlystopping_cb, mdchcheckpoint_cb],
  9 )
  □ Epoch 1/50
  2/22 [=====] - 1303s 40s/step - loss: 1.8962 - accuracy: 0.4436 - val_loss: 0.7012 - val_accuracy: 0.6924
  Epoch 2/50
  2/22 [=====] - 38s 2s/step - loss: 0.4995 - accuracy: 0.8209 - val_loss: 0.4632 - val_accuracy: 0.8172
  Epoch 3/50
  2/22 [=====] - 37s 2s/step - loss: 0.4543 - accuracy: 0.8289 - val_loss: 0.3637 - val_accuracy: 0.8484
  Epoch 4/50
  2/22 [=====] - 34s 1s/step - loss: 0.3028 - accuracy: 0.8890 - val_loss: 0.2594 - val_accuracy: 0.8900
  Epoch 5/50
  2/22 [=====] - 33s 1s/step - loss: 0.2978 - accuracy: 0.8942 - val_loss: 0.3043 - val_accuracy: 0.8975
  Epoch 6/50
  2/22 [=====] - 37s 2s/step - loss: 0.2188 - accuracy: 0.9143 - val_loss: 0.1984 - val_accuracy: 0.9376
  Epoch 7/50
  2/22 [=====] - 33s 1s/step - loss: 0.2507 - accuracy: 0.9053 - val_loss: 0.3124 - val_accuracy: 0.8811
  Epoch 8/50
  2/22 [=====] - 38s 2s/step - loss: 0.4209 - accuracy: 0.8478 - val_loss: 0.3394 - val_accuracy: 0.8782
  Epoch 9/50
  2/22 [=====] - 33s 1s/step - loss: 0.2860 - accuracy: 0.8931 - val_loss: 0.2101 - val_accuracy: 0.9287
  Epoch 10/50
  2/22 [=====] - 33s 1s/step - loss: 0.2024 - accuracy: 0.9243 - val_loss: 0.3187 - val_accuracy: 0.8841
  Epoch 11/50
  2/22 [=====] - 33s 1s/step - loss: 0.1845 - accuracy: 0.9321 - val_loss: 0.4560 - val_accuracy: 0.8574
  Epoch 12/50
  2/22 [=====] - 37s 2s/step - loss: 0.2411 - accuracy: 0.9120 - val_loss: 0.4115 - val_accuracy: 0.8484
  Epoch 13/50
  2/22 [=====] - 37s 2s/step - loss: 0.1728 - accuracy: 0.9343 - val_loss: 0.3662 - val_accuracy: 0.8975
  Epoch 14/50
  2/22 [=====] - 33s 1s/step - loss: 0.1451 - accuracy: 0.9491 - val_loss: 0.2965 - val_accuracy: 0.9183
  Epoch 15/50
  2/22 [=====] - 33s 1s/step - loss: 0.2361 - accuracy: 0.9143 - val_loss: 0.2500 - val_accuracy: 0.9183
  Epoch 16/50
  2/22 [=====] - 33s 1s/step - loss: 0.1580 - accuracy: 0.9380 - val_loss: 0.2450 - val_accuracy: 0.9094
  [10] 1 print(model.evaluate(valid_ds))
  2/22 [=====] - 12s 503ms/step - loss: 0.1984 - accuracy: 0.9376
  [0.19843372702598572, 0.9375928640365601]
  <> 1 #redefine BATCH_SIZE, por si la cantidad de muestras es menor a 128
  2 BATCH_SIZE = 128
  3

```

Ilustración 86: Métricas para la prueba 32.

Resumen

- VALID_SPLIT = 0.2
- EPOCHS = 50
- Modelo estable en 16 EPOCHS, 1825 segs (equivalente a 30.41 minutos)
- loss: 0.1984 - accuracy: 0.9376
- GPU

```

rnd [ 68 69 123 53 112 100 62 122 13 97]
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: VictoriaFolosaPaz Predicted: VictoriaFolosaPaz
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer

```

Ilustración 87: Resultados para la prueba 32.

```

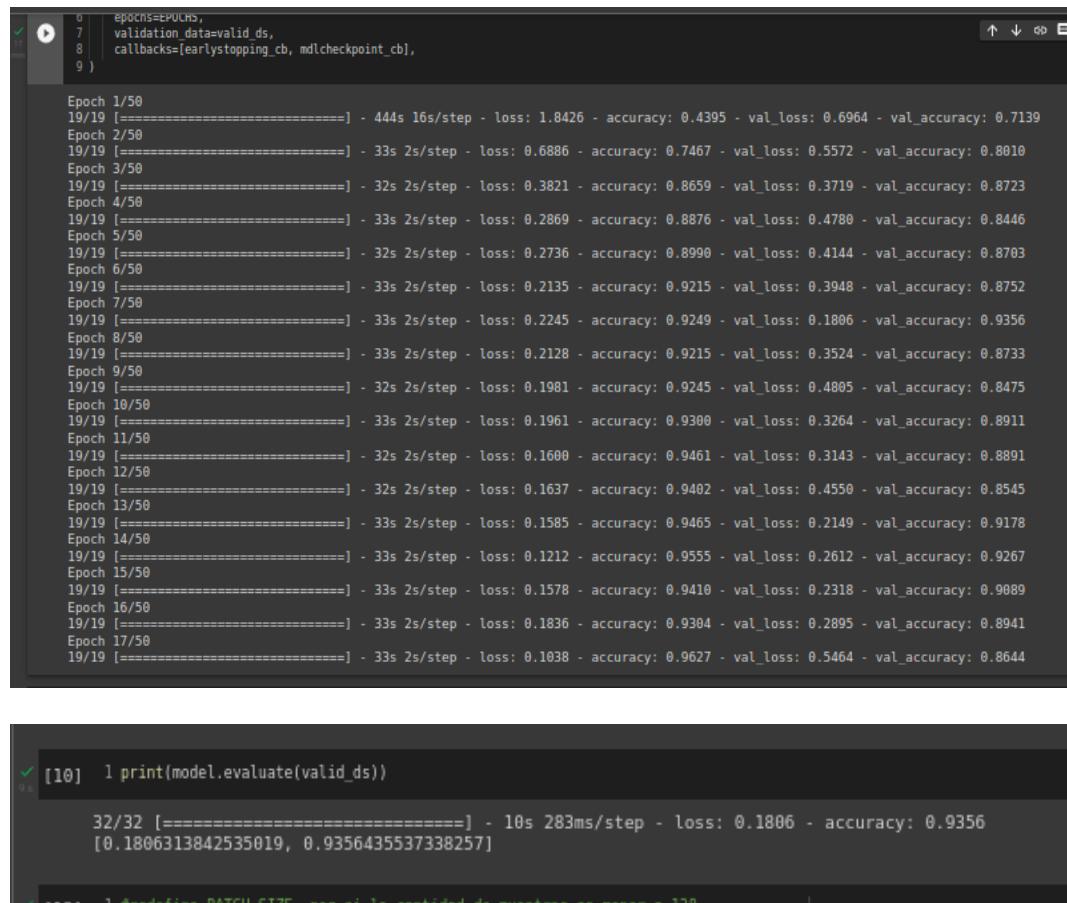
proba.cercania=[0.9955007433891296] orden_categoria=49 nombre_categoria=MariaEugeniaVidal
predicción(Voz en Vivo) = MariaEugeniaVidal ... y_pred=[49] index=133
registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [49]

```

Ilustración 88: Predicción para la prueba 32.

Prueba 33 – voces género Femenino, split 0.3

- procesamiento_V0020104.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.3
- EPOCHS = 50



The image shows two screenshots of a Jupyter Notebook interface. The top screenshot displays a code cell with a checkmark indicating successful execution. The code defines a model's training parameters (epochs, validation data, callbacks) and then prints a log of training epochs from 1 to 19. The log includes metrics like loss and accuracy for both training and validation sets. The bottom screenshot shows another code cell with a checkmark, printing the results of a model's evaluation on a validation dataset. It shows 32/32 steps completed in 10s, with a loss of 0.1806 and an accuracy of 0.9356. A warning message at the bottom of this cell indicates that the BATCH_SIZE was redefined to 128.

```

0 |   epochs=EPOCHS,
1 |   validation_data=valid_ds,
2 |   callbacks=[earlystopping_cb, mdlcheckpoint_cb],
3 |
4 | Epoch 1/50
5 | 19/19 [=====] - 444s 16s/step - loss: 1.8426 - accuracy: 0.4395 - val_loss: 0.6964 - val_accuracy: 0.7139
6 | Epoch 2/50
7 | 19/19 [=====] - 33s 2s/step - loss: 0.6886 - accuracy: 0.7467 - val_loss: 0.5572 - val_accuracy: 0.8010
8 | Epoch 3/50
9 | 19/19 [=====] - 32s 2s/step - loss: 0.3821 - accuracy: 0.8659 - val_loss: 0.3719 - val_accuracy: 0.8723
10 | Epoch 4/50
11 | 19/19 [=====] - 33s 2s/step - loss: 0.2869 - accuracy: 0.8876 - val_loss: 0.4780 - val_accuracy: 0.8446
12 | Epoch 5/50
13 | 19/19 [=====] - 32s 2s/step - loss: 0.2736 - accuracy: 0.8900 - val_loss: 0.4144 - val_accuracy: 0.8703
14 | Epoch 6/50
15 | 19/19 [=====] - 33s 2s/step - loss: 0.2135 - accuracy: 0.9215 - val_loss: 0.3948 - val_accuracy: 0.8752
16 | Epoch 7/50
17 | 19/19 [=====] - 33s 2s/step - loss: 0.2245 - accuracy: 0.9249 - val_loss: 0.1806 - val_accuracy: 0.9356
18 | Epoch 8/50
19 | 19/19 [=====] - 33s 2s/step - loss: 0.2128 - accuracy: 0.9215 - val_loss: 0.3524 - val_accuracy: 0.8733
20 | Epoch 9/50
21 | 19/19 [=====] - 32s 2s/step - loss: 0.1981 - accuracy: 0.9245 - val_loss: 0.4805 - val_accuracy: 0.8475
22 | Epoch 10/50
23 | 19/19 [=====] - 33s 2s/step - loss: 0.1961 - accuracy: 0.9300 - val_loss: 0.3264 - val_accuracy: 0.8911
24 | Epoch 11/50
25 | 19/19 [=====] - 32s 2s/step - loss: 0.1600 - accuracy: 0.9461 - val_loss: 0.3143 - val_accuracy: 0.8891
26 | Epoch 12/50
27 | 19/19 [=====] - 32s 2s/step - loss: 0.1637 - accuracy: 0.9402 - val_loss: 0.4550 - val_accuracy: 0.8545
28 | Epoch 13/50
29 | 19/19 [=====] - 33s 2s/step - loss: 0.1585 - accuracy: 0.9465 - val_loss: 0.2149 - val_accuracy: 0.9178
30 | Epoch 14/50
31 | 19/19 [=====] - 33s 2s/step - loss: 0.1212 - accuracy: 0.9555 - val_loss: 0.2612 - val_accuracy: 0.9267
32 | Epoch 15/50
33 | 19/19 [=====] - 33s 2s/step - loss: 0.1578 - accuracy: 0.9410 - val_loss: 0.2318 - val_accuracy: 0.9089
34 | Epoch 16/50
35 | 19/19 [=====] - 33s 2s/step - loss: 0.1836 - accuracy: 0.9304 - val_loss: 0.2895 - val_accuracy: 0.8941
36 | Epoch 17/50
37 | 19/19 [=====] - 33s 2s/step - loss: 0.1038 - accuracy: 0.9627 - val_loss: 0.5464 - val_accuracy: 0.8644

```

```

[10] 1 print(model.evaluate(valid_ds))

32/32 [=====] - 10s 283ms/step - loss: 0.1806 - accuracy: 0.9356
[0.1806313842535019, 0.9356435537338257]

# Redefine BATCH_SIZE por si la cantidad de muestra es menor a 128

```

Ilustración 89: Métricas para la prueba 33.

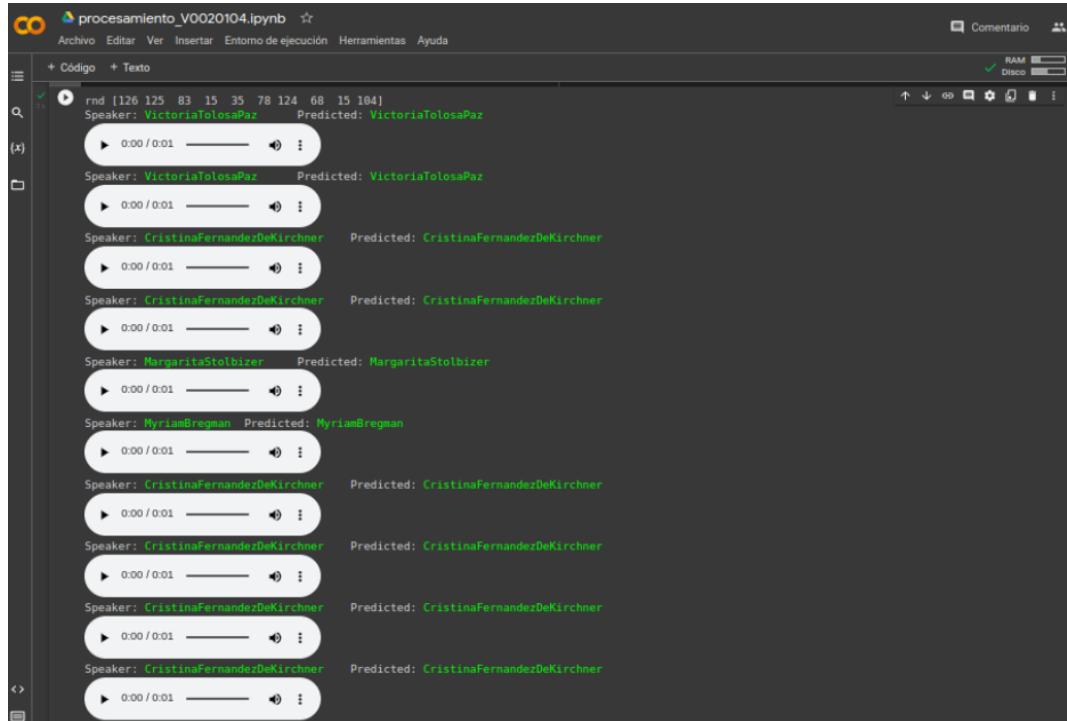


Ilustración 90: Resultados para la prueba 33.



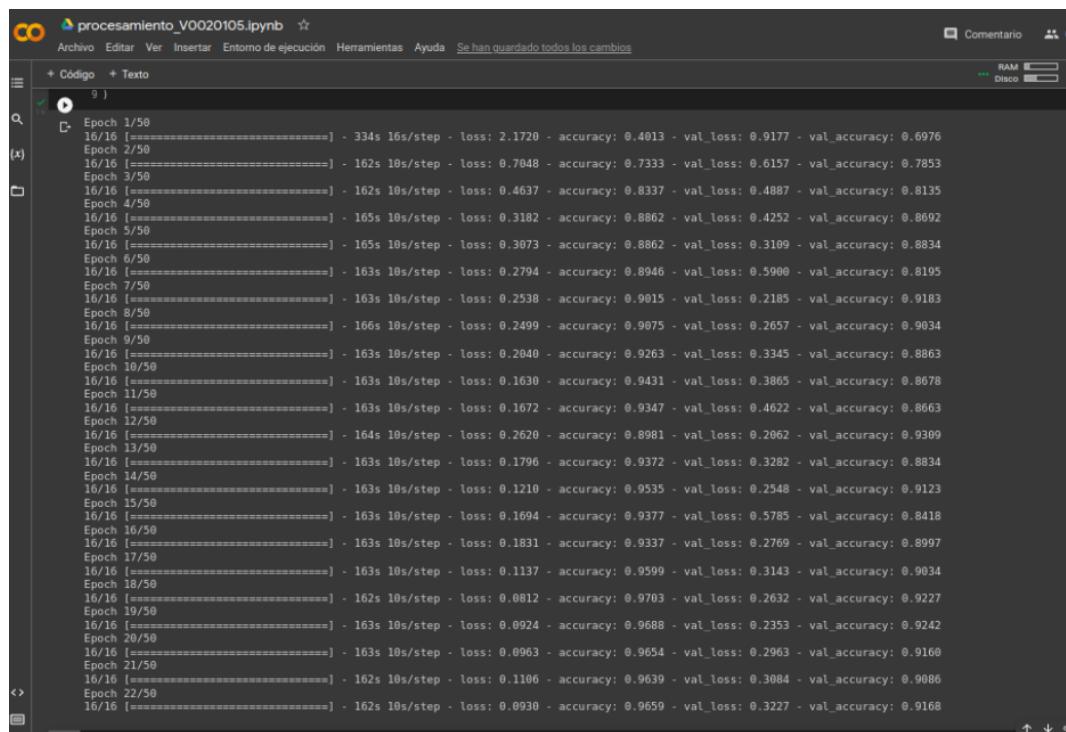
Ilustración 91: Métricas para la prueba 33.

Resumen

- VALID_SPLIT = 0.3
- EPOCHS = 50
- ⇒ alcancé modelo estable en 17 EPOCHS, en 967 segs (16.11 minutos)
- loss: 0.1806 - accuracy: 0.9356
- GPU

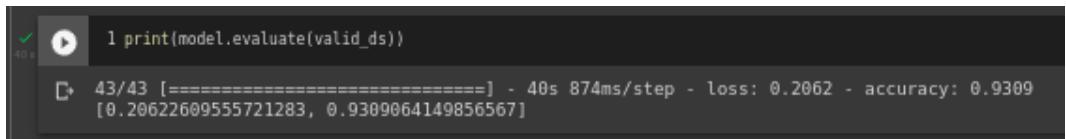
Prueba 34 – voces género Femenino, split 0.4

- procesamiento_V0020104.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.4
- EPOCHS = 50



The screenshot shows the Jupyter Notebook interface with the file 'procesamiento_V0020105.ipynb' open. The code cell contains a single line of Python code: 'print(model.evaluate(valid_ds))'. The output pane displays the results of 50 epochs of training. Each epoch summary includes the epoch number, training progress bar, step duration, loss value, and accuracy values for both training and validation sets. The final output shows a test accuracy of 0.9309.

```
1 print(model.evaluate(valid_ds))
2
3 40 s
4
5 1 Epoch 1/50
6 16/16 [=====] - 334s 16s/step - loss: 2.1720 - accuracy: 0.4013 - val_loss: 0.9177 - val_accuracy: 0.6976
7 Epoch 2/50
8 16/16 [=====] - 162s 10s/step - loss: 0.7048 - accuracy: 0.7333 - val_loss: 0.6157 - val_accuracy: 0.7853
9 Epoch 3/50
10 16/16 [=====] - 162s 10s/step - loss: 0.4637 - accuracy: 0.8337 - val_loss: 0.4887 - val_accuracy: 0.8135
11 Epoch 4/50
12 16/16 [=====] - 165s 10s/step - loss: 0.3182 - accuracy: 0.8862 - val_loss: 0.4252 - val_accuracy: 0.8692
13 Epoch 5/50
14 16/16 [=====] - 165s 10s/step - loss: 0.3073 - accuracy: 0.8862 - val_loss: 0.3109 - val_accuracy: 0.8834
15 Epoch 6/50
16 16/16 [=====] - 163s 10s/step - loss: 0.2794 - accuracy: 0.8946 - val_loss: 0.5900 - val_accuracy: 0.8195
17 Epoch 7/50
18 16/16 [=====] - 163s 10s/step - loss: 0.2538 - accuracy: 0.9015 - val_loss: 0.2185 - val_accuracy: 0.9183
19 Epoch 8/50
20 16/16 [=====] - 166s 10s/step - loss: 0.2499 - accuracy: 0.9075 - val_loss: 0.2657 - val_accuracy: 0.9034
21 Epoch 9/50
22 16/16 [=====] - 163s 10s/step - loss: 0.2040 - accuracy: 0.9263 - val_loss: 0.3345 - val_accuracy: 0.8863
23 Epoch 10/50
24 16/16 [=====] - 163s 10s/step - loss: 0.1630 - accuracy: 0.9431 - val_loss: 0.3865 - val_accuracy: 0.8678
25 Epoch 11/50
26 16/16 [=====] - 163s 10s/step - loss: 0.1672 - accuracy: 0.9347 - val_loss: 0.4622 - val_accuracy: 0.8663
27 Epoch 12/50
28 16/16 [=====] - 164s 10s/step - loss: 0.2620 - accuracy: 0.8981 - val_loss: 0.2062 - val_accuracy: 0.9309
29 Epoch 13/50
30 16/16 [=====] - 163s 10s/step - loss: 0.1796 - accuracy: 0.9372 - val_loss: 0.3282 - val_accuracy: 0.8834
31 Epoch 14/50
32 16/16 [=====] - 163s 10s/step - loss: 0.1210 - accuracy: 0.9535 - val_loss: 0.2548 - val_accuracy: 0.9123
33 Epoch 15/50
34 16/16 [=====] - 163s 10s/step - loss: 0.1694 - accuracy: 0.9377 - val_loss: 0.3785 - val_accuracy: 0.8418
35 Epoch 16/50
36 16/16 [=====] - 163s 10s/step - loss: 0.1831 - accuracy: 0.9337 - val_loss: 0.2769 - val_accuracy: 0.8997
37 Epoch 17/50
38 16/16 [=====] - 163s 10s/step - loss: 0.1137 - accuracy: 0.9599 - val_loss: 0.3143 - val_accuracy: 0.9034
39 Epoch 18/50
40 16/16 [=====] - 162s 10s/step - loss: 0.0812 - accuracy: 0.9703 - val_loss: 0.2632 - val_accuracy: 0.9227
41 Epoch 19/50
42 16/16 [=====] - 163s 10s/step - loss: 0.0924 - accuracy: 0.9688 - val_loss: 0.2353 - val_accuracy: 0.9242
43 Epoch 20/50
44 16/16 [=====] - 163s 10s/step - loss: 0.0963 - accuracy: 0.9654 - val_loss: 0.2963 - val_accuracy: 0.9160
45 Epoch 21/50
46 16/16 [=====] - 162s 10s/step - loss: 0.1106 - accuracy: 0.9639 - val_loss: 0.3084 - val_accuracy: 0.9086
47 Epoch 22/50
48 16/16 [=====] - 162s 10s/step - loss: 0.0930 - accuracy: 0.9659 - val_loss: 0.3227 - val_accuracy: 0.9168
```



The screenshot shows the Jupyter Notebook interface with the file 'procesamiento_V0020105.ipynb' open. The code cell contains a single line of Python code: 'print(model.evaluate(valid_ds))'. The output pane displays the results of 43 epochs of testing. The final output shows a test accuracy of 0.9309.

```
1 print(model.evaluate(valid_ds))
2
3 40 s
4
5 1 43/43 [=====] - 40s 874ms/step - loss: 0.2062 - accuracy: 0.9309
6 [0.20622609555721283, 0.9309064149856567]
```

Ilustración 92: Métricas para la prueba 34.

```

rnd [114 24 121 23 45 47 25 72 66 79]
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: MarisEugeniaVidal Predicted: MariaEugeniaVidal
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer

```

Ilustración 93: Resultados para la prueba 34.

```

C: MariaEugeniaVidal
1) proba.cercanía=[0.98451828956604] orden_categoria=49 nombre_categoria=MariaEugeniaVidal
2) predicción(Voz en Vivo) = MariaEugeniaVidal ... y_pred=[49] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [49]

```

Ilustración 94: Métricas para la prueba 34.

Resumen

- VALID_SPLIT = 0.4
- EPOCHS = 50
- ⇒ alcancé modelo estable en 22 EPOCHS, en 1100 segs (18.33 minutos)
- loss: 0.9309 - accuracy: 0.2062
- GPU

Prueba 35 – voces género Femenino, split 0.6

- procesamiento_V0020106.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.6
- EPOCHS = 50

The screenshot shows the Google Colab interface for the notebook 'procesamiento_V0020106.ipynb'. The main area displays a long list of training logs from epoch 1 to 50. The logs include metrics like loss and accuracy for both training and validation sets. The right side of the interface features a 'Recursos' panel with graphs for RAM usage, GPU usage, and disk activity over time. The bottom status bar indicates the execution completed at 18:08.

```
1/11 [=====] - 33s 3s/step - loss: 0.1935 - accuracy: 0.9287 - val_loss: 0.2694
1/11 [=====] - 33s 3s/step - loss: 0.1273 - accuracy: 0.9517 - val_loss: 0.5212 - val_accuracy: 0.8485
Epoch 17/50
1/11 [=====] - 33s 3s/step - loss: 0.1336 - accuracy: 0.9525 - val_loss: 0.4434 - val_accuracy: 0.8579
Epoch 18/50
1/11 [=====] - 33s 3s/step - loss: 0.1500 - accuracy: 0.9465 - val_loss: 0.3965 - val_accuracy: 0.8842
Epoch 19/50
1/11 [=====] - 33s 3s/step - loss: 0.1282 - accuracy: 0.9555 - val_loss: 0.2944 - val_accuracy: 0.9109
Epoch 20/50
1/11 [=====] - 33s 3s/step - loss: 0.1810 - accuracy: 0.9332 - val_loss: 0.2515 - val_accuracy: 0.9153
Epoch 21/50
1/11 [=====] - 34s 3s/step - loss: 0.1293 - accuracy: 0.9569 - val_loss: 0.3697 - val_accuracy: 0.8777
Epoch 22/50
1/11 [=====] - 33s 3s/step - loss: 0.0991 - accuracy: 0.9629 - val_loss: 0.2564 - val_accuracy: 0.9252
Epoch 23/50
1/11 [=====] - 32s 3s/step - loss: 0.0843 - accuracy: 0.9725 - val_loss: 0.3222 - val_accuracy: 0.9974
Epoch 24/50
1/11 [=====] - 33s 3s/step - loss: 0.1060 - accuracy: 0.9614 - val_loss: 0.2509 - val_accuracy: 0.9282
Epoch 25/50
1/11 [=====] - 33s 3s/step - loss: 0.1360 - accuracy: 0.9480 - val_loss: 0.2786 - val_accuracy: 0.9074
Epoch 26/50
1/11 [=====] - 33s 3s/step - loss: 0.1172 - accuracy: 0.9629 - val_loss: 0.2040 - val_accuracy: 0.9307
Epoch 27/50
1/11 [=====] - 33s 3s/step - loss: 0.0794 - accuracy: 0.9710 - val_loss: 0.3562 - val_accuracy: 0.9069
Epoch 28/50
1/11 [=====] - 33s 3s/step - loss: 0.1215 - accuracy: 0.9562 - val_loss: 0.2912 - val_accuracy: 0.9178
Epoch 29/50
1/11 [=====] - 33s 3s/step - loss: 0.0934 - accuracy: 0.9659 - val_loss: 0.3270 - val_accuracy: 0.9074
Epoch 30/50
1/11 [=====] - 33s 3s/step - loss: 0.0936 - accuracy: 0.9696 - val_loss: 0.3256 - val_accuracy: 0.9168
Epoch 31/50
1/11 [=====] - 33s 3s/step - loss: 0.0631 - accuracy: 0.9733 - val_loss: 0.3243 - val_accuracy: 0.9178
Epoch 32/50
1/11 [=====] - 33s 3s/step - loss: 0.0711 - accuracy: 0.9777 - val_loss: 0.3727 - val_accuracy: 0.8941
Epoch 33/50
1/11 [=====] - 33s 3s/step - loss: 0.0595 - accuracy: 0.9770 - val_loss: 0.4232 - val_accuracy: 0.8921
Epoch 34/50
1/11 [=====] - 33s 3s/step - loss: 0.0475 - accuracy: 0.9866 - val_loss: 0.3647 - val_accuracy: 0.9168
Epoch 35/50
1/11 [=====] - 33s 3s/step - loss: 0.0473 - accuracy: 0.9866 - val_loss: 0.2995 - val_accuracy: 0.9218
Epoch 36/50
1/11 [=====] - 33s 3s/step - loss: 0.0496 - accuracy: 0.9822 - val_loss: 0.2679 - val_accuracy: 0.9411
```

The screenshot shows the Jupyter Notebook terminal output for cell [10]. It displays the command 'print(model.evaluate(valid_ds))' and its resulting output, which shows a single batch of 64 samples with a loss of 0.2040 and an accuracy of 0.9307.

```
[10]: 1 print(model.evaluate(valid_ds))

64/64 [=====] - 19s 293ms/step - loss: 0.2040 - accuracy: 0.9307
[0.20403175055980682, 0.9306930899620056]
```

Ilustración 95: Métricas para la prueba 35.

Ilustración 96: Métricas para la prueba 35.

Ilustración 97: Predicción para la prueba 35.

Resumen

- VALID_SPLIT = 0.6
- EPOCHS = 50
- ⇒ alcancé modelo estable en 37 EPOCHS, en 2463 segs (41.05 minutos)
- loss: 0.2040 - accuracy: 0.9307
- GPU

Prueba 36 – voces género Femenino, split 0.7

Esperando por recursos. No puedo conectarme con Colab.

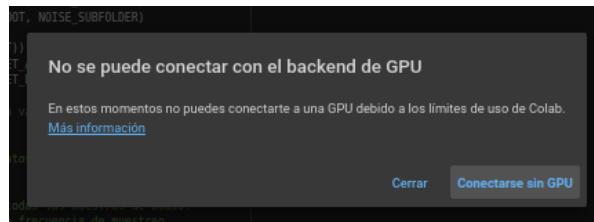


Ilustración 98: Issue al tomar recursos desde Google Colab.

Luego de haber solucionado el problema anterior,

```
1) 2) 3) 4) 5) 6) 7) 8) 9) 10) 11) 12) 13) 14) 15) 16) 17) 18) 19) 20) 21) 22) 23) 24) 25) 26) 27) 28) 29) 30) 31) 32) 33)
```

```
+ Código + Texto
```

```
Epoch 13/50
Epoch 14/50
Epoch 15/50
Epoch 16/50
Epoch 17/50
Epoch 18/50
Epoch 19/50
Epoch 20/50
Epoch 21/50
Epoch 22/50
Epoch 23/50
Epoch 24/50
Epoch 25/50
Epoch 26/50
Epoch 27/50
Epoch 28/50
Epoch 29/50
Epoch 30/50
Epoch 31/50
Epoch 32/50
Epoch 33/50
```

```
[ ] 1 print(model.evaluate(valid_ds))
```

Recursos X

RAM del backend de Google Compute Engine que utiliza Python 3

Mostrando recursos desde las 19:19

RAM del sistema Disco

Gestionar sesiones Cambiar tipo de entorno de ejecución

✓ 1 h 14 min 55 s - completado a las 20:36

```
1 print(model.evaluate(valid_ds))
```

```
74/74 [=====] - 59s 788ms/step - loss: 0.2493 - accuracy: 0.9232
[0.24929171800613403, 0.9231748580932617]
```

Ilustración 99: Métricas para la prueba 36.

```

rnd [ 95 87 127  9 90 83 78 22 79 99]
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: VictoriaTeossaPaz Predicted: VictoriaTeossaPaz
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵

```

Ilustración 100: Resultados para la prueba 36.

```

MariaEugeniaVidal
1) proba.cercania=[0.9998952150344849] orden_categoria=49 nombre_categoria=MariaEugeniaVidal
2) predicción(Voz en Vivo) = MariaEugeniaVidal ... y_pred=[49] index=133
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [49]
▶ 0:00 / 0:01 ━━━━━━ ⏪ ⏴ ⏵

```

Ilustración 101: Predicción para la prueba 36.

Resumen

- VALID_SPLIT = 0.7
- EPOCHS = 50
- ⇒ alcancé modelo estable en 33 EPOCHS, en 4162 segs (69.37 minutos)
- loss: 0.2493 - accuracy: 0.9232

Prueba 37 – voces género Femenino, split 0.8

- procesamiento_V0020106.ipynb
- df = prueba_006.xls

- VALID_SPLIT = 0.8

- EPOCHS = 50

```

+ Código + Texto
epoch=50,
validation_data=valid_ds,
callbacks=[earlystopping_cb, mdtcheckpoint_cb],

```

Epoch 1/50
 Epoch 2/50
 Epoch 3/50
 Epoch 4/50
 Epoch 5/50
 Epoch 6/50
 Epoch 7/50
 Epoch 8/50
 Epoch 9/50
 Epoch 10/50
 Epoch 11/50
 Epoch 12/50
 Epoch 13/50
 Epoch 14/50
 Epoch 15/50
 Epoch 16/50
 Epoch 17/50
 Epoch 18/50
 Epoch 19/50
 Epoch 20/50
 Epoch 21/50
 Epoch 22/50
 Epoch 23/50
 Epoch 24/50
 Epoch 25/50
 Epoch 26/50
 Epoch 27/50
 Epoch 28/50
 Epoch 29/50
 Epoch 30/50
 Epoch 31/50
 Epoch 32/50
 Epoch 33/50
 Epoch 34/50
 Epoch 35/50
 Epoch 36/50
 Epoch 37/50
 Epoch 38/50
 Epoch 39/50
 Epoch 40/50
 Epoch 41/50
 Epoch 42/50
 Epoch 43/50
 Epoch 44/50
 Epoch 45/50
 Epoch 46/50
 Epoch 47/50
 Epoch 48/50
 Epoch 49/50
 Epoch 50/50

[12]: l print(model.evaluate(valid_ds))

31 min 16 s completado a las 20:16

```

[12]: l print(model.evaluate(valid_ds))

```

85/85 [=====] - 49s 571ms/step - loss: 0.4750 - accuracy: 0.8288
 [0.4750221073627472, 0.8288154602050781]

Ilustración 102: Métricas para la prueba 37.

```

rnd(36, 1, 16, 104, 43, 7, 53, 64, 117, 15)
Speaker: VictoriaTolosaPaz Predicted: VictoriaTolosaPaz
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
Speaker: VictoriaTolosaPaz Predicted: MyriamBregman
Speaker: VictoriaTolosaPaz Predicted: CristinaFernandezDeKirchner
Speaker: MariaEugenioVidal Predicted: MariaEugenioVidal
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: MyriamBregman Predicted: MyriamBregman
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
Speaker: MariaEugenioVidal Predicted: MariaEugenioVidal
Speaker: VictoriaTolosaPaz Predicted: VictoriaTolosaPaz

```

Ilustración 103: Resultados para la prueba 37.

```

proba.cercania=[0.9849360585212708] orden_categoria=49 nombre_categoria=MariaEugenioVidal
prediccion(Voz en Vivo) = MariaEugenioVidal ... y_pred=[49] index=133
registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [49]

```

Ilustración 104: Métricas para la prueba 37.

Resumen

- VALID_SPLIT = 0.8
- EPOCHS = 50
- Modelo estable en 37 EPOCHS, en 1838 segs (equivalente a 30.6 minutos)
- loss: 0.4750 - accuracy: 0.8288

Prueba 38 – voces género Femenino, split 0.9

- procesamiento_V0020106.ipynb
- df = prueba_006.xls

- VALID_SPLIT = 0.9
- EPOCHS = 50

```
[1] 1 print(model.evaluate(valid_ds))
[1] 1 min
[1] 1 1 h 14 min 37 s - completado a las 21:02
```

Epoch 14/50
Epoch 15/50
Epoch 16/50
Epoch 17/50
Epoch 18/50
Epoch 19/50
Epoch 20/50
Epoch 21/50
Epoch 22/50
Epoch 23/50
Epoch 24/50
Epoch 25/50
Epoch 26/50
Epoch 27/50
Epoch 28/50
Epoch 29/50
Epoch 30/50
Epoch 31/50
Epoch 32/50
Epoch 33/50
Epoch 34/50
Epoch 35/50

3/3 [=====] - 98s 44s/step - loss: 0.4119 - accuracy: 0.8368 - val_loss: 0.5115 - val_accuracy: 0.7049
3/3 [=====] - 96s 43s/step - loss: 0.3714 - accuracy: 0.8724 - val_loss: 0.7729 - val_accuracy: 0.7281
3/3 [=====] - 97s 43s/step - loss: 0.3232 - accuracy: 0.9050 - val_loss: 0.6671 - val_accuracy: 0.7729
3/3 [=====] - 97s 43s/step - loss: 0.2521 - accuracy: 0.9880 - val_loss: 0.7959 - val_accuracy: 0.7812
3/3 [=====] - 97s 43s/step - loss: 0.1999 - accuracy: 0.9880 - val_loss: 1.1915 - val_accuracy: 0.7168
3/3 [=====] - 97s 43s/step - loss: 0.2037 - accuracy: 0.9228 - val_loss: 1.0844 - val_accuracy: 0.7502
3/3 [=====] - 97s 43s/step - loss: 0.2556 - accuracy: 0.8813 - val_loss: 0.7065 - val_accuracy: 0.8050
3/3 [=====] - 98s 43s/step - loss: 0.2249 - accuracy: 0.9199 - val_loss: 0.9762 - val_accuracy: 0.7512
3/3 [=====] - 98s 44s/step - loss: 0.1640 - accuracy: 0.9288 - val_loss: 0.6144 - val_accuracy: 0.8386
3/3 [=====] - 97s 43s/step - loss: 0.1495 - accuracy: 0.9377 - val_loss: 0.8858 - val_accuracy: 0.7957
3/3 [=====] - 98s 44s/step - loss: 0.1325 - accuracy: 0.9466 - val_loss: 0.5934 - val_accuracy: 0.8581
3/3 [=====] - 100s 45s/step - loss: 0.1455 - accuracy: 0.9347 - val_loss: 0.4366 - val_accuracy: 0.8769
3/3 [=====] - 99s 44s/step - loss: 0.1278 - accuracy: 0.9703 - val_loss: 0.5960 - val_accuracy: 0.8317
3/3 [=====] - 97s 43s/step - loss: 0.1380 - accuracy: 0.9466 - val_loss: 0.4686 - val_accuracy: 0.8624
3/3 [=====] - 98s 44s/step - loss: 0.0818 - accuracy: 0.9822 - val_loss: 0.8670 - val_accuracy: 0.8168
3/3 [=====] - 98s 43s/step - loss: 0.0807 - accuracy: 0.9674 - val_loss: 0.7834 - val_accuracy: 0.8231
3/3 [=====] - 100s 45s/step - loss: 0.0580 - accuracy: 0.9911 - val_loss: 1.4656 - val_accuracy: 0.7452
3/3 [=====] - 100s 45s/step - loss: 0.1029 - accuracy: 0.9703 - val_loss: 1.1989 - val_accuracy: 0.7848
3/3 [=====] - 99s 44s/step - loss: 0.0348 - accuracy: 0.9881 - val_loss: 0.7895 - val_accuracy: 0.8518
3/3 [=====] - 97s 43s/step - loss: 0.0549 - accuracy: 0.9852 - val_loss: 0.6189 - val_accuracy: 0.8726
3/3 [=====] - 98s 44s/step - loss: 0.0517 - accuracy: 0.9822 - val_loss: 0.8929 - val_accuracy: 0.8373
3/3 [=====] - 98s 44s/step - loss: 0.0397 - accuracy: 0.9852 - val_loss: 0.7054 - val_accuracy: 0.8667

```
[11] 1 print(model.evaluate(valid_ds))
[11] 1 min
[11] 1 95/95 [=====] - 77s 794ms/step - loss: 0.4366 - accuracy: 0.8769
[11] 1 [0.4366236627101898, 0.8768976926803589]
```

Ilustración 105: Métricas para la prueba 38.

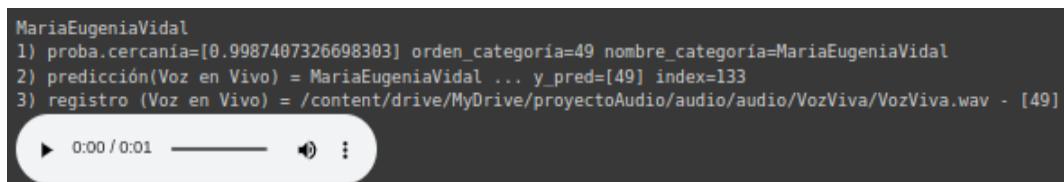


Ilustración 106: Predicción para la prueba 38.

Ilustración 107: Resultados para la prueba 38.

Resumen

- VALID_SPLIT = 0.9
- EPOCHS = 50
- Modelo estable en 35 EPOCHS, en 3637 segs (equivalente a 60.6 minutos)
- loss: 0.4366 - accuracy: 0.8769

Prueba 39 – voces género Femenino, split 0.25

- procesamiento_V0020110.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.25
- EPOCHS = 50

```

procesamiento_V0020110.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
Q 20/20 [=====] - 194s 9s/step - loss: 0.2522 - accuracy: 0.0125 - val_loss: 0.5207 - val_accuracy: 0.8264
(x) Epoch 6/50
D 20/20 [=====] - 195s 9s/step - loss: 0.2771 - accuracy: 0.0022 - val_loss: 0.4850 - val_accuracy: 0.8264
20/20 [=====] - 196s 9s/step - loss: 0.2605 - accuracy: 0.0018 - val_loss: 0.4557 - val_accuracy: 0.8478
Epoch 8/50
20/20 [=====] - 195s 9s/step - loss: 0.2018 - accuracy: 0.9256 - val_loss: 0.4321 - val_accuracy: 0.8478
Epoch 9/50
20/20 [=====] - 195s 9s/step - loss: 0.1930 - accuracy: 0.9359 - val_loss: 0.2743 - val_accuracy: 0.9001
Epoch 10/50
20/20 [=====] - 195s 9s/step - loss: 0.1671 - accuracy: 0.9359 - val_loss: 0.2860 - val_accuracy: 0.8977
Epoch 12/50
20/20 [=====] - 196s 9s/step - loss: 0.1437 - accuracy: 0.9497 - val_loss: 0.2897 - val_accuracy: 0.9096
Epoch 20/20 [=====] - 195s 9s/step - loss: 0.1635 - accuracy: 0.9378 - val_loss: 0.2361 - val_accuracy: 0.9191
Epoch 13/50
20/20 [=====] - 194s 9s/step - loss: 0.1272 - accuracy: 0.9525 - val_loss: 0.2687 - val_accuracy: 0.9073
Epoch 14/50
20/20 [=====] - 195s 9s/step - loss: 0.1305 - accuracy: 0.9509 - val_loss: 0.2240 - val_accuracy: 0.9275
Epoch 15/50
20/20 [=====] - 194s 9s/step - loss: 0.1677 - accuracy: 0.9390 - val_loss: 0.3811 - val_accuracy: 0.8621
Epoch 16/50
20/20 [=====] - 194s 9s/step - loss: 0.1758 - accuracy: 0.9347 - val_loss: 0.2286 - val_accuracy: 0.9239
Epoch 17/50
20/20 [=====] - 194s 9s/step - loss: 0.1208 - accuracy: 0.9584 - val_loss: 0.1714 - val_accuracy: 0.9394
Epoch 18/50
20/20 [=====] - 195s 9s/step - loss: 0.0997 - accuracy: 0.9612 - val_loss: 0.2555 - val_accuracy: 0.9239
Epoch 19/50
20/20 [=====] - 194s 9s/step - loss: 0.1119 - accuracy: 0.9604 - val_loss: 0.2580 - val_accuracy: 0.9191
Epoch 20/50
20/20 [=====] - 198s 10s/step - loss: 0.0887 - accuracy: 0.9683 - val_loss: 0.2048 - val_accuracy: 0.9405
Epoch 21/50
20/20 [=====] - 195s 9s/step - loss: 0.0811 - accuracy: 0.9723 - val_loss: 0.2049 - val_accuracy: 0.9358
Epoch 22/50
20/20 [=====] - 195s 9s/step - loss: 0.0988 - accuracy: 0.9636 - val_loss: 0.2556 - val_accuracy: 0.9346
Epoch 23/50
20/20 [=====] - 194s 9s/step - loss: 0.0916 - accuracy: 0.9663 - val_loss: 0.2333 - val_accuracy: 0.9322
Epoch 24/50
20/20 [=====] - 195s 9s/step - loss: 0.0700 - accuracy: 0.9747 - val_loss: 0.2811 - val_accuracy: 0.9168
Epoch 25/50
20/20 [=====] - 195s 9s/step - loss: 0.0876 - accuracy: 0.9667 - val_loss: 0.2471 - val_accuracy: 0.9287
Epoch 26/50
20/20 [=====] - 195s 9s/step - loss: 0.0694 - accuracy: 0.9731 - val_loss: 0.1950 - val_accuracy: 0.9465
Epoch 27/50
20/20 [=====] - 195s 9s/step - loss: 0.0752 - accuracy: 0.9731 - val_loss: 0.2955 - val_accuracy: 0.9203
20/20 [=====] - 195s 9s/step - loss: 0.0752 - accuracy: 0.9731 - val_loss: 0.2955 - val_accuracy: 0.9203

```

✓ 1 h 32 min 37 s completado a las 22:43

```

[11] 1 print(model.evaluate(valid_ds))

27/27 [=====] - 23s 790ms/step - loss: 0.1714 - accuracy: 0.9394
[0.1713695377111435, 0.9393579363822937]

```

Ilustración 108: Métricas para la prueba 39.

```

procesamiento_V0020110.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda RAM Disco Editar
+ Código + Texto
Q rnd [ 72 22 39 62 82 19 24 102 127 10]
(x) Speaker: MyriamBregman Predicted: MyriamBregman
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: MyriamBregman Predicted: CristinaFernandezDeKirchner
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: MyriamBregman Predicted: MyriamBregman
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: VictoriaTolosaPaz Predicted: VictoriaTolosaPaz
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
D ► 0:00/0:01 ━━━━ ◁ ━━ ; 
Speaker: MyriamBregman Predicted: MyriamBregman
D ► 0:00/0:01 ━━━━ ◁ ━━ ;

```

✓ 6 s completado a las 22:59

Ilustración 109: Resultados para la prueba 39.

```
C:\ MariaEugenioVidal
 1) proba.cercania=[0.9987945556640625] orden_categoria=49 nombre_categoria=MariaEugenioVidal
 2) predicción(Voz en Vivo) = MariaEugenioVidal ... y_pred=[49] index=133
 3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [49]
```

Ilustración 110: Métricas para la prueba 39.

Resumen

- VALID_SPLIT = 0.25
- EPOCHS = 50
- Modelo estable en 27 EPOCHS, en 5502 segs (equivalente a 91.7 minutos)
- loss: 0.1714 - accuracy: 0.9394

Prueba 40 – voces género Femenino, split 0.35

- procesamiento_V0020111.ipynb
- df = prueba_006.xls
- VALID_SPLIT = 0.35
- EPOCHS = 50

```

procesamiento_V002011.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
1 #obrescribo EPOCHS = 1
2 EPOCHS = 5
3
4 history = model.fit(
5     train_ds,
6     epochs=EPOCHS,
7     validation_data=valid_ds,
8     callbacks=[earlystopping_cb, mdlcheckpoint_cb],
9 )

```

Epoch 1/50
18/18 [=====] - 454s 19s/step - loss: 2.3386 - accuracy: 0.3741 - val_loss: 0.8147 - val_accuracy: 0.6927
Epoch 2/50
18/18 [=====] - 173s 9s/step - loss: 0.7956 - accuracy: 0.7104 - val_loss: 0.9022 - val_accuracy: 0.7109
Epoch 3/50
18/18 [=====] - 172s 9s/step - loss: 0.4780 - accuracy: 0.8250 - val_loss: 0.4677 - val_accuracy: 0.8149
Epoch 4/50
18/18 [=====] - 175s 9s/step - loss: 0.3245 - accuracy: 0.8785 - val_loss: 0.4589 - val_accuracy: 0.8447
Epoch 5/50
18/18 [=====] - 173s 9s/step - loss: 0.3431 - accuracy: 0.8744 - val_loss: 0.4834 - val_accuracy: 0.8421
Epoch 6/50
18/18 [=====] - 174s 9s/step - loss: 0.3136 - accuracy: 0.8926 - val_loss: 0.3644 - val_accuracy: 0.8658
Epoch 7/50
18/18 [=====] - 177s 10s/step - loss: 0.3082 - accuracy: 0.8922 - val_loss: 0.2205 - val_accuracy: 0.9270
Epoch 8/50
18/18 [=====] - 175s 9s/step - loss: 0.3052 - accuracy: 0.8945 - val_loss: 0.3716 - val_accuracy: 0.8973
Epoch 9/50
18/18 [=====] - 176s 9s/step - loss: 0.2764 - accuracy: 0.9041 - val_loss: 0.2696 - val_accuracy: 0.9126
Epoch 10/50
18/18 [=====] - 173s 9s/step - loss: 0.2378 - accuracy: 0.9187 - val_loss: 0.3924 - val_accuracy: 0.8616
Epoch 11/50
18/18 [=====] - 173s 9s/step - loss: 0.2022 - accuracy: 0.9251 - val_loss: 0.2737 - val_accuracy: 0.9015
Epoch 12/50
18/18 [=====] - 172s 9s/step - loss: 0.2221 - accuracy: 0.9178 - val_loss: 0.3036 - val_accuracy: 0.8990
Epoch 13/50
18/18 [=====] - 172s 9s/step - loss: 0.2845 - accuracy: 0.9214 - val_loss: 0.4033 - val_accuracy: 0.8523
Epoch 14/50
18/18 [=====] - 171s 9s/step - loss: 0.1655 - accuracy: 0.9342 - val_loss: 0.2452 - val_accuracy: 0.9168
Epoch 15/50
18/18 [=====] - 171s 9s/step - loss: 0.2087 - accuracy: 0.9278 - val_loss: 0.4090 - val_accuracy: 0.9480
Epoch 16/50
18/18 [=====] - 172s 9s/step - loss: 0.1770 - accuracy: 0.9356 - val_loss: 0.3413 - val_accuracy: 0.8752
Epoch 17/50
18/18 [=====] - 172s 9s/step - loss: 0.1406 - accuracy: 0.9497 - val_loss: 0.3611 - val_accuracy: 0.8862

```

[ ] 1 print(model.evaluate(valid_ds))

```

58 min 53 s completado a las 22:15

```

[10] 1 print(model.evaluate(valid_ds))

37/37 [=====] - 32s 821ms/step - loss: 0.2205 - accuracy: 0.9270
[0.220450846091079712, 0.9269949197769165]

```

Ilustración 111: Métricas para la prueba 40.

```

procesamiento_V002011.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
1 34 | | | display(Audio(audios[index, :, :].squeeze(), rate=SAMPLING_RATE))

```

Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: CristinaFernandezDeKirchner Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: VictoriaTolosaPaz Predicted: VictoriaTolosaPaz
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: CristinaFernandezDeKirchner Predicted: CristinaFernandezDeKirchner
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: MyriamBregman Predicted: MyriamBregman
▶ 0:00/0:01 ━━━━ ↷ ━
Speaker: MargaritaStolbizer Predicted: MargaritaStolbizer
▶ 0:00/0:01 ━━━━ ↷ ━

8 s completado a las 22:42

Ilustración 112: Resultados para la prueba 40.

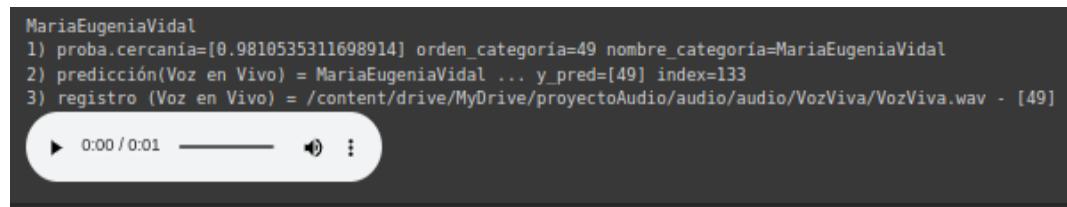


Ilustración 113: Métricas para la prueba 40.

- VALID_SPLIT = 0.35
- EPOCHS = 50
- Modelo estable en 17 EPOCHS, en 3225 segs (equivalente a 53.75 minutos)
- loss: 0.2204 - accuracy: 0.9269

NOTEBOOK	VALID_SPLIT	EPOCHS	USED_EPOCHS	ELAPSED_TIME	MEAN_EPOCH_TIME	accuracy	loss	Backend
procesamiento_V002_0101.ipynb	0,1	50	16	3448	215,5	0,9494	0,1525	TPU
procesamiento_V002_0103.ipynb	0,2	50	16	1825	114,1	0,9376	0,1984	GPU
procesamiento_V002_0110.ipynb	0,25	50	27	5502	203,7	0,9394	0,1714	-
procesamiento_V002_0104.ipynb	0,3	50	17	967	56,8	0,9356	0,1806	GPU
procesamiento_V002_0111.ipynb	0,35	50	17	3225	189,7	0,9269	0,2204	TPU
procesamiento_V002_0105.ipynb	0,4	50	22	3760	62,7	0,9309	0,2062	TPU
procesamiento_V002_0102.ipynb	0,5	50	37	4532	122,5	0,9483	0,1667	TPU
procesamiento_V002_0106.ipynb	0,6	50	36	2463	68,4	0,9307	0,2040	GPU
procesamiento_V002_0107.ipynb	0,7	50	33	4162	126,1	0,9232	0,2493	-
procesamiento_V002_0108.ipynb	0,8	50	20	1838	91,9	0,8288	0,4750	-
procesamiento_V002_0109.ipynb	0,9	50	35	3637	103,9	0,8769	0,4366	-

Tabla 6: “Métricas Accuracy / Loss” vs. “valid_split / EPOCHS”

Prueba 41 – Voz de un imitador vs. Voz real

Para aportar resultados confiables, durante la prueba se tomaron archivos de audio de un hablante (Cristina Fernández de Kirchner - voz actual y voz de hace 20 años), y también se entrenó al modelo con archivos de audio de su imitador (Fátima Florez)

Al realizar la predicción, el modelo pudo diferenciar con predicciones del orden del 98,5% si la voz era compatible con el imitador o con la voz real.

Prueba 42 – Deep Fake

La voz generada mediante Inteligencia Artificial (Deep Fake) se utilizó para entrenar el modelo y predecir si el mismo es vulnerable a este tipo de engaño.

El conjunto de datos para el hablante “*Donald Trump*”, alcanzó la cantidad de archivos de voz de la persona real esperada.

Métricas obtenidas: *Loss: 0.089 - accuracy: 0.9674*. Sin embargo, el modelo predijo que la voz artificial era compatible con alta certeza con la voz real de Donald Trump.

Prueba 43 – Voz generada por AI

Esta prueba de Deep Fake, se alimenta con la voz de “**VAL KILMER**” y su voz generada con INTELIGENCIA ARTIFICIAL.

Voz artificial - 1

- https://www.youtube.com/watch?v=Fahwy_sBPtQ
- https://youtu.be/Fahwy_sBPtQ
- Audio desde minuto 14:04 ~ hasta minuto 15:24

Voz artificial – 2

- <https://www.youtube.com/watch?v=OSMue60Gg6s>
- <https://youtu.be/OSMue60Gg6s>
- Audio desde minuto 00:00 ~ hasta minuto 01:49

Voz original en la película TOP GUN 1

- <https://www.youtube.com/watch?v=zq8rHTcveyA>
- <https://youtu.be/zq8rHTcveyA>
- Audio desde minuto 00:44 ~ hasta minuto 00:46

VAL KILMER última voz real

- <https://www.youtube.com/watch?v=6FamkvVLG9Y>
- <https://youtu.be/6FamkvVLG9Y>
- Audio desde minuto 08:35 ~ hasta minuto 08:40

```

+ Código + Texto
[2] 16
[15]
1 #-----
2 #
3 # Inicialización de constantes y variables
4 #
5 #
6 #C MAIN_AUDIO_PATH = '/home/jmmiguez/proyectoAudio/audio/'
7 C_MAIN_AUDIO_PATH = '/content/drive/MyDrive/proyectoAudio/audio/'
8
9 #C_DATASET_AUDIO_PATH = "/home/jmmiguez/proyectoAudio/audio/Downloads/"
10 C_DATASET_AUDIO_PATH = "/content/drive/MyDrive/proyectoAudio/audio/Downloads/"
11
12 V_NOMBRE_ARCHIVO_MAESTRO = 'audio_maestro'
13 V_PATH_Y_NOMBRE_ARCHIVO_MP4 = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.mp4'
14 V_PATH_Y_NOMBRE_ARCHIVO_WAV = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'
15
16 #V_DETALLE_VIDEO_LINK = 'VAL KILMER y su nueva voz generada con INTELIGENCIA ARTIFICIAL'
17 V_VIDEO_LINK = 'https://youtu.be/zq8rHtcveyA'
18 V_LABELS DESDE EXCEL = 'ValKilmer.xls'
19 V_PATH_Y_NOMBRE_ARCHIVO_XLS = C_MAIN_AUDIO_PATH + V_LABELS DESDE EXCEL
20 #V_CANT_LABELS_EXCEL = 1
21 V_CHUNK_LENGTH = 1000

```

Ilustración 114: Preprocesamiento para la prueba 43.

	A	B	C
1	segs_ini	segs_fin	Nombre_Senador
2	44	46	ValKilmerAI
3	1	2	RicardoLopezMurphy
4	1	2	MartinLousteau
5			

Tabla 10: Hablantes que se procesan en la prueba 43

- procesamiento_V0020201.ipynb
- df = prueba_007.xls
- VALID_SPLIT = 0.3 y GPU
- EPOCHS = 50

```

procesamiento_V0020201.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
[✓] 5/5 [=====] - 8s ls/step - loss: 0.0881 - accuracy: 0.9833 - val_loss: 0.0265 - val_accuracy: 0.9957
Epoch 10/50
5/5 [=====] - 8s ls/step - loss: 0.0467 - accuracy: 0.9833 - val_loss: 0.0062 - val_accuracy: 0.9957
Epoch 11/50
5/5 [=====] - 8s ls/step - loss: 0.0179 - accuracy: 0.9981 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 12/50
5/5 [=====] - 8s ls/step - loss: 0.0211 - accuracy: 0.9963 - val_loss: 0.0123 - val_accuracy: 0.9957
Epoch 13/50
5/5 [=====] - 8s ls/step - loss: 0.0202 - accuracy: 0.9944 - val_loss: 0.0052 - val_accuracy: 0.9957
Epoch 14/50
5/5 [=====] - 8s ls/step - loss: 0.0247 - accuracy: 0.9926 - val_loss: 0.0034 - val_accuracy: 1.0000
Epoch 15/50
5/5 [=====] - 8s ls/step - loss: 0.0036 - accuracy: 0.9981 - val_loss: 0.0035 - val_accuracy: 1.0000
Epoch 16/50
5/5 [=====] - 8s ls/step - loss: 0.0179 - accuracy: 0.9926 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 17/50
5/5 [=====] - 8s ls/step - loss: 9.9346e-04 - accuracy: 1.0000 - val_loss: 0.0057 - val_accuracy: 0.9957
Epoch 18/50
5/5 [=====] - 8s ls/step - loss: 0.0024 - accuracy: 0.9981 - val_loss: 0.0038 - val_accuracy: 1.0000
Epoch 19/50
5/5 [=====] - 8s ls/step - loss: 2.2651e-04 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 20/50
5/5 [=====] - 8s ls/step - loss: 7.8333e-04 - accuracy: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 21/50
5/5 [=====] - 8s ls/step - loss: 4.7138e-04 - accuracy: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 22/50
5/5 [=====] - 8s ls/step - loss: 1.3197e-04 - accuracy: 1.0000 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 23/50
5/5 [=====] - 8s ls/step - loss: 5.8230e-05 - accuracy: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 24/50
5/5 [=====] - 8s ls/step - loss: 4.2327e-05 - accuracy: 1.0000 - val_loss: 0.0024 - val_accuracy: 1.0000
Epoch 25/50
5/5 [=====] - 8s ls/step - loss: 3.2958e-05 - accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 26/50
5/5 [=====] - 8s ls/step - loss: 4.5112e-05 - accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 27/50
5/5 [=====] - 8s ls/step - loss: 1.3515e-05 - accuracy: 1.0000 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 28/50
5/5 [=====] - 8s ls/step - loss: 1.1738e-05 - accuracy: 1.0000 - val_loss: 0.0025 - val_accuracy: 1.0000
Epoch 29/50
5/5 [=====] - 8s ls/step - loss: 0.0077 - accuracy: 0.9963 - val_loss: 0.0061 - val_accuracy: 0.9957
Epoch 30/50
5/5 [=====] - 8s ls/step - loss: 0.0255 - accuracy: 0.9981 - val_loss: 0.0057 - val_accuracy: 1.0000
[ ] 1 print(model.evaluate(valid_ds))

```

RAM del sistema

RAM de la GPU

Disco

Gestionar sesiones

Cambiar tipo de entorno de ejecución

```

procesamiento_V0020201.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
[✓] 1 print(model.evaluate(valid_ds))
8/8 [=====] - 2s 230ms/step - loss: 0.0018 - accuracy: 1.0000
[0.0017829957650974393, 1.0]

```

Ilustración 115: Métricas para la prueba 43.

```

procesamiento_V0020201.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
[✓] Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: ValKilmerAI Predicted: ValKilmerAI
Speaker: MartinLousteau Predicted: MartinLousteau
[ ] 1 #

```

RAM del sistema

RAM de la GPU

Disco

Gestionar sesiones

Cambiar tipo de entorno de ejecución

Ilustración 116: Resultados para la prueba 43.

```

C:\> MartinLousteau
1) proba.cercanía=[0.95735764503479] orden_categoria=122 nombre_categoria=MartinLousteau
2) predicción(Voz en Vivo) = MartinLousteau ... y_pred=[122] index=135
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [122]

```

Ilustración 117: Predicción para la prueba 43.

Predijo erróneamente:

- Las voces no estaban en el mismo idioma.
- El modelo estaba sobreentrenado.
- La voz viva no se usó para entrenamiento ni para la validación. Las voces difieren en años temporalmente.

Prueba 44 – Voz generada por AI

MorganFreemanAI

- <https://www.youtube.com/watch?v=F4G6GNFz0O8>
- <https://youtu.be/F4G6GNFz0O8>
- Audio desde minuto 00:00 ~ hasta minuto 00:47

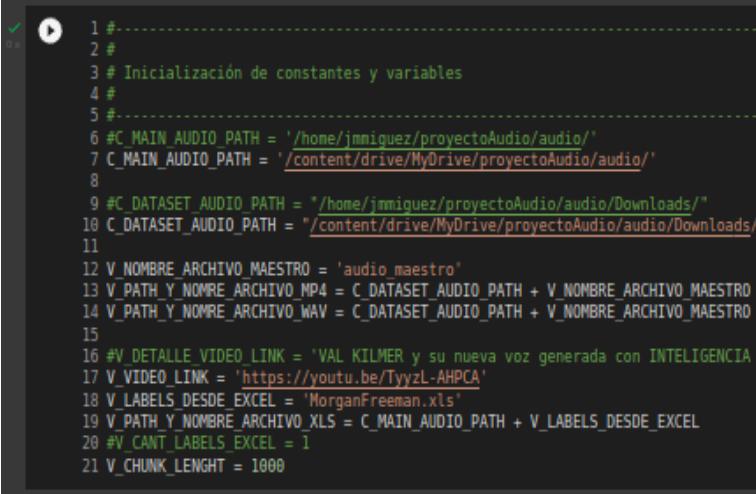
MorganFreeman

- <https://www.youtube.com/watch?v=TyyzL-AHPCA>
- <https://youtu.be/TyyzL-AHPCA>
- Audio desde minuto 00:19 ~ hasta minuto 00:34

Configuración

- procesamiento_V0020202.ipynb
- df = prueba_007.xls

- VALID_SPLIT = 0.3 y GPU
- EPOCHS = 50

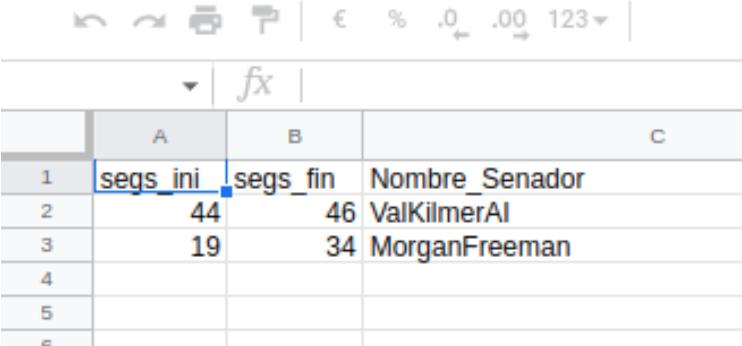


```

1 #
2 #
3 # Inicialización de constantes y variables
4 #
5 #
6 #C_MAIN_AUDIO_PATH = '/home/jmmiguez/proyectoAudio/audio/'
7 C_MAIN_AUDIO_PATH = '/content/drive/MyDrive/proyectoAudio/audio/'
8
9 #C_DATASET_AUDIO_PATH = "/home/jmmiguez/proyectoAudio/audio/Downloads/"
10 C_DATASET_AUDIO_PATH = "/content/drive/MyDrive/proyectoAudio/audio/Downloads/"
11
12 V_NOMBRE_ARCHIVO_MAESTRO = 'audio_maestro'
13 V_PATH_Y_NOMBRE_ARCHIVO_MP4 = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.mp4'
14 V_PATH_Y_NOMBRE_ARCHIVO_WAV = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'
15
16 #V_DETALLE_VIDEO_LINK = 'VAL KILMER y su nueva voz generada con INTELIGENCIA ARTIFICIAL'
17 V_VIDEO_LINK = 'https://youtu.be/TyyzL-AHPCA'
18 V_LABELS_DESDE_EXCEL = 'MorganFreeman.xls'
19 V_PATH_Y_NOMBRE_ARCHIVO_XLS = C_MAIN_AUDIO_PATH + V_LABELS_DESDE_EXCEL
20 #V_CANT_LABELS_EXCEL = 1
21 V_CHUNK_LENGTH = 1000

```

Ilustración 118: Preprocesamiento para la prueba 44.



	A	B	C
1	segs_ini	segs_fin	Nombre_Senador
2	44	46	ValKilmerAI
3	19	34	MorganFreeman
4			
5			
R			

Tabla 11: Hablantes que se procesan en la prueba 44

- procesamiento_V0020202.ipynb
- df = prueba_008.xls
- VALID_SPLIT = 0.2 y GPU
- EPOCHS = 50

```

procesamiento_V0020202.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
1/1 [=====] - ls 1s/step - loss: 0.2436 - accuracy: 0.9153 - val_loss: 0.2973 - val_accuracy: 0.9260
Epoch 31/50
1/1 [=====] - ls 1s/step - loss: 0.0680 - accuracy: 0.9661 - val_loss: 0.1424 - val_accuracy: 0.9286
Epoch 32/50
1/1 [=====] - ls 1s/step - loss: 0.1051 - accuracy: 0.9492 - val_loss: 0.1185 - val_accuracy: 0.9286
Epoch 33/50
1/1 [=====] - ls 1s/step - loss: 0.2075 - accuracy: 0.9322 - val_loss: 0.1426 - val_accuracy: 0.9286
Epoch 34/50
1/1 [=====] - ls 1s/step - loss: 0.0949 - accuracy: 0.9661 - val_loss: 0.1770 - val_accuracy: 0.9286
Epoch 35/50
1/1 [=====] - ls 1s/step - loss: 0.1480 - accuracy: 0.9661 - val_loss: 0.1860 - val_accuracy: 0.9286
Epoch 36/50
1/1 [=====] - ls 1s/step - loss: 0.1364 - accuracy: 0.9322 - val_loss: 0.1310 - val_accuracy: 0.9286
Epoch 37/50
1/1 [=====] - ls 1s/step - loss: 0.0844 - accuracy: 0.9661 - val_loss: 0.0867 - val_accuracy: 0.9286
Epoch 38/50
1/1 [=====] - 2s/step - loss: 0.0538 - accuracy: 1.0000 - val_loss: 0.0572 - val_accuracy: 1.0000
Epoch 39/50
1/1 [=====] - ls 1s/step - loss: 0.0638 - accuracy: 0.9661 - val_loss: 0.0463 - val_accuracy: 1.0000
Epoch 40/50
1/1 [=====] - ls 1s/step - loss: 0.0647 - accuracy: 0.9661 - val_loss: 0.0567 - val_accuracy: 1.0000
Epoch 41/50
1/1 [=====] - ls 1s/step - loss: 0.0314 - accuracy: 1.0000 - val_loss: 0.0909 - val_accuracy: 0.9286
Epoch 42/50
1/1 [=====] - ls 1s/step - loss: 0.0374 - accuracy: 1.0000 - val_loss: 0.1260 - val_accuracy: 0.9286
Epoch 43/50
1/1 [=====] - ls 1s/step - loss: 0.0265 - accuracy: 1.0000 - val_loss: 0.1692 - val_accuracy: 0.9286
Epoch 44/50
1/1 [=====] - ls 1s/step - loss: 0.0364 - accuracy: 0.9831 - val_loss: 0.1332 - val_accuracy: 0.9286
Epoch 45/50
1/1 [=====] - ls 1s/step - loss: 0.0128 - accuracy: 1.0000 - val_loss: 0.1062 - val_accuracy: 0.9286
Epoch 46/50
1/1 [=====] - ls 1s/step - loss: 0.0871 - accuracy: 1.0899 - val_loss: 0.0992 - val_accuracy: 0.9286
Epoch 47/50
1/1 [=====] - ls 1s/step - loss: 0.0447 - accuracy: 0.9831 - val_loss: 0.0991 - val_accuracy: 1.0000
Epoch 48/50
1/1 [=====] - ls 1s/step - loss: 0.0134 - accuracy: 1.0000 - val_loss: 0.0817 - val_accuracy: 1.0000
Epoch 49/50
1/1 [=====] - ls 1s/step - loss: 0.1110 - accuracy: 0.9661 - val_loss: 0.0859 - val_accuracy: 1.0000
Epoch 50/50
1/1 [=====] - ls 1s/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.0413 - val_accuracy: 1.0000
[ ] 1 print(model.evaluate(valid_ds))
[ ] 2 #redondeo BATCH_SIZE por si la cantidad de muestras es menor a 178
2 min 1 s  completado a las 0:31

```

```

1 print(model.evaluate(valid_ds))

1/1 [=====] - 0s 271ms/step - loss: 0.0413 - accuracy: 1.0000
[0.041260119527578354, 1.0]

```

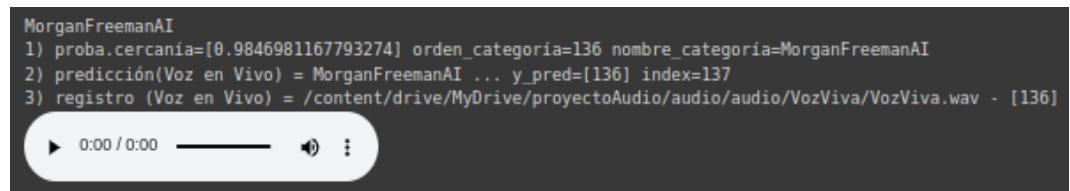
Ilustración 119: Métricas para la prueba 44.

```

procesamiento_V0020202.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
14 | display(Audio(audios[index, :, :].squeeze(), rate=SAMPLING_RATE))
rnd [7 8 1 1 6 4 6 8 6 0]
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: ValKilmerAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: ValKilmerAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: ValKilmerAI Predicted: MorganFreemanAI
▶ 0.00/0.01 ━━━━ ◁ ━━
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0.00/0.01 ━━━━ ◁ ━━
[ ] 1 s  completado a las 0:33

```

Ilustración 120: Resultados para la prueba 44.



```
MorganFreemanAI
1) proba.cercania=[0.9846981167793274] orden_categoria=136 nombre_categoria=MorganFreemanAI
2) predicción(Voz en Vivo) = MorganFreemanAI ... y_pred=[136] index=137
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [136]
```

The terminal window shows the output of an AI model's prediction. It includes the predicted category (MorganFreemanAI), the predicted probability (0.9846981167793274), the predicted index (136), and the path to the recorded audio file (VozViva.wav). Below the terminal window is a media player interface with a play button, a progress bar at 0:00 / 0:00, a volume icon, and a settings icon.

Ilustración 121: Predicciones para la prueba 44.

Son muy pocos datos para el entrenamiento.

Predijo la Voz de Val Kilmer original (sin AI) como la voz de Morgan Freeman.

Prueba 45 – Voz generada por AI, Split 0.5

- procesamiento_V0020203.ipynb
- df = prueba_008.xls
- VALID_SPLIT = 0.5 y GPU
- EPOCHS = 50

Usando 37 archivos para entrenamiento.

Usando 36 archivos para validación.

```

procesamiento_V0020203.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
RAM Disco Comentario Compartir Editar Recursos X ...
(GPU del backend de Google Compute Engine que utiliza Python 3 Mostrando recursos desde las 0:37
RAM del sistema
RAM de la GPU
Disco
(x) Epoch 27/50
1/1 [=====] - ls ls/step - loss: 0.1417 - accuracy: 0.9730 - val_loss: 0.0710 - val_accuracy: 0.9722
Epoch 28/50
1/1 [=====] - ls ls/step - loss: 0.0420 - accuracy: 1.0000 - val_loss: 0.0614 - val_accuracy: 0.9722
Epoch 29/50
1/1 [=====] - ls ls/step - loss: 0.1248 - accuracy: 0.9730 - val_loss: 0.0935 - val_accuracy: 0.9722
Epoch 30/50
1/1 [=====] - ls ls/step - loss: 0.0355 - accuracy: 1.0000 - val_loss: 0.1278 - val_accuracy: 0.9722
Epoch 31/50
1/1 [=====] - ls ls/step - loss: 0.1240 - accuracy: 0.9459 - val_loss: 0.0665 - val_accuracy: 0.9722
Epoch 32/50
1/1 [=====] - ls ls/step - loss: 0.0147 - accuracy: 1.0000 - val_loss: 0.0273 - val_accuracy: 1.0000
Epoch 33/50
1/1 [=====] - ls ls/step - loss: 0.0143 - accuracy: 1.0000 - val_loss: 0.0135 - val_accuracy: 1.0000
Epoch 34/50
1/1 [=====] - ls ls/step - loss: 0.1833 - accuracy: 0.9730 - val_loss: 0.0352 - val_accuracy: 0.9722
Epoch 35/50
1/1 [=====] - ls ls/step - loss: 0.0315 - accuracy: 1.0000 - val_loss: 0.0793 - val_accuracy: 0.9722
Epoch 36/50
1/1 [=====] - ls ls/step - loss: 0.0433 - accuracy: 0.9730 - val_loss: 0.0914 - val_accuracy: 0.9722
Epoch 37/50
1/1 [=====] - ls ls/step - loss: 0.0629 - accuracy: 0.9730 - val_loss: 0.0438 - val_accuracy: 0.9722
Epoch 38/50
1/1 [=====] - ls ls/step - loss: 0.0130 - accuracy: 1.0000 - val_loss: 0.0139 - val_accuracy: 1.0000
Epoch 39/50
1/1 [=====] - ls ls/step - loss: 0.0182 - accuracy: 1.0000 - val_loss: 0.0106 - val_accuracy: 1.0000
Epoch 40/50
1/1 [=====] - ls ls/step - loss: 0.0190 - accuracy: 1.0000 - val_loss: 0.0409 - val_accuracy: 0.9722
Epoch 41/50
1/1 [=====] - ls ls/step - loss: 0.0100 - accuracy: 1.0000 - val_loss: 0.0882 - val_accuracy: 0.9722
Epoch 42/50
1/1 [=====] - ls ls/step - loss: 0.0059 - accuracy: 1.0000 - val_loss: 0.1299 - val_accuracy: 0.9722
Epoch 43/50
1/1 [=====] - ls ls/step - loss: 0.0115 - accuracy: 1.0000 - val_loss: 0.1286 - val_accuracy: 0.9722
Epoch 44/50
1/1 [=====] - ls ls/step - loss: 0.0095 - accuracy: 1.0000 - val_loss: 0.0928 - val_accuracy: 0.9722
Epoch 45/50
1/1 [=====] - ls ls/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.0593 - val_accuracy: 0.9722
Epoch 46/50
1/1 [=====] - ls ls/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0299 - val_accuracy: 0.9722
Epoch 47/50
1/1 [=====] - ls ls/step - loss: 4.1981e-04 - accuracy: 1.0000 - val_loss: 0.0126 - val_accuracy: 1.0000
Epoch 48/50
1/1 [=====] - ls ls/step - loss: 0.0155 - accuracy: 1.0000 - val_loss: 0.0458 - val_accuracy: 0.9722
Epoch 49/50
1/1 [=====] - ls ls/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.0899 - val_accuracy: 0.9722

```

```

1 print(model.evaluate(valid_ds))
2/2 [=====] - ls 12ms/step - loss: 0.0106 - accuracy: 1.0000
[0.010610654018819332, 1.0]

```

Ilustración 122: Métricas para la prueba 45.

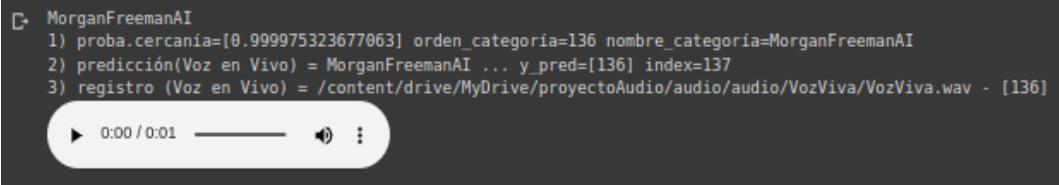
```

rnd [4 1 2 9 2 8 6 9 4 2]
Speaker: ValKilmerAI Predicted: ValKilmerAI
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
Speaker: ValKilmerAI Predicted: ValKilmerAI
Speaker: ValKilmerAI Predicted: ValKilmerAI
Speaker: ValKilmerAI Predicted: ValKilmerAI
Speaker: MorganFreemanAI Predicted: MorganFreemanAI
Speaker: ValKilmerAI Predicted: ValKilmerAI

```

Ilustración 123: Resultados para la prueba 45.

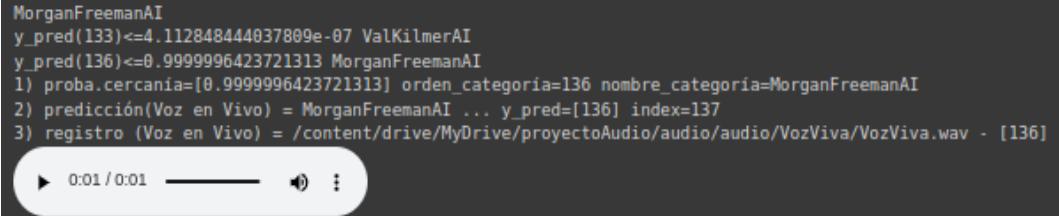
La voz de Morgan Freeman, se predice como MorganFreemanAI



```
MorganFreemanAI
1) proba.cercania=[0.999975323677063] orden_categoria=136 nombre_categoria=MorganFreemanAI
2) predicción(Voz en Vivo) = MorganFreemanAI ... y_pred=[136] index=137
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [136]
```

Ilustración 124: Predicción 1 para la prueba 45.

La voz de ValKilmer original, se predice como MorganFreemanAI



```
MorganFreemanAI
y_pred(133)<=4.112848444037809e-07 ValKilmerAI
y_pred(136)<=0.9999996423721313 MorganFreemanAI
1) proba.cercania=[0.9999996423721313] orden_categoria=136 nombre_categoria=MorganFreemanAI
2) predicción(Voz en Vivo) = MorganFreemanAI ... y_pred=[136] index=137
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [136]
```

Ilustración 125: Predicción 2 para la prueba 45.

Poco entrenamiento, Voz alejada de las voces entrenadas.

Devolvió alta probabilidad de cercanía, de manera equivocada.

Conclusión: No se puede avanzar con un modelo con pocas muestras de entrenamiento. La cantidad de audios de entrenamiento debe ser más amplia.

Prueba 46 – Voz generada por AI

Se agregó la voz de Donald Trump Original.

Se realizó la predicción con la voz de Donald Trumpo con AI.

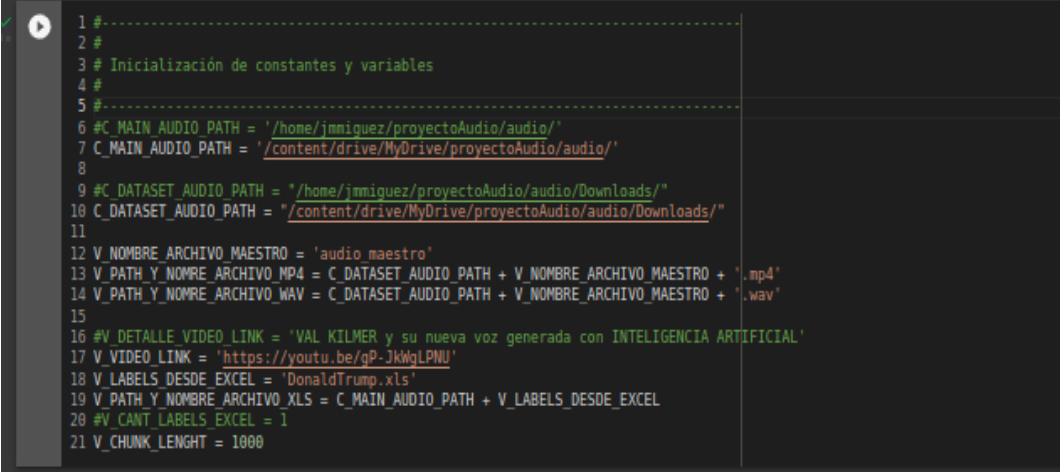
DonaldTrump

- <https://www.youtube.com/watch?v=gP-JkWgLPNU>
- <https://youtu.be/gP-JkWgLPNU>

- Audio desde minuto 00:00 ~ hasta minuto 09:15

DonaldTrumpAI

- <https://www.youtube.com/watch?v=aPp5lcqgISk>
- <https://youtu.be/aPp5lcqgISk>
- Audio desde minuto 00:00 ~ hasta minuto 00:26



```

1 #-----#
2 #
3 # Inicialización de constantes y variables
4 #
5 #-----#
6 #C_MAIN_AUDIO_PATH = '/home/jmmiguez/proyectoAudio/audio/'
7 C_MAIN_AUDIO_PATH = '/content/drive/MyDrive/proyectoAudio/audio/'
8
9 #C_DATASET_AUDIO_PATH = '/home/jmmiguez/proyectoAudio/audio/Downloads/'
10 C_DATASET_AUDIO_PATH = '/content/drive/MyDrive/proyectoAudio/audio/Downloads/'
11
12 V_NOMBRE_ARCHIVO_MAESTRO = 'audio maestro'
13 V_PATH_Y_NOMBRE_ARCHIVO_MP4 = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.mp4'
14 V_PATH_Y_NOMBRE_ARCHIVO_WAV = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'
15
16 #V_DETALLE_VIDEO_LINK = 'VAL KILMER y su nueva voz generada con INTELIGENCIA ARTIFICIAL'
17 V_VIDEO_LINK = 'https://youtu.be/gP-JkWqLPNU'
18 V_LABELS_DESDE_EXCEL = 'DonaldTrump.xls'
19 V_PATH_Y_NOMBRE_ARCHIVO_XLS = C_MAIN_AUDIO_PATH + V_LABELS_DESDE_EXCEL
20 #V_CANT_LABELS_EXCEL = 1
21 V_CHUNK_LENGTH = 1000

```

Ilustración 126: Preprocesamiento para la prueba 46.

- procesamiento_V0020204.ipynb
- df = prueba_009.xls
- VALID_SPLIT = 0.2 y GPU
- EPOCHS = 50

prueba_009.xls XLSX Guardado

Archivo Editar Ver Insertar Formato Datos

A5

	A	B	C
1	segs_ini	segs_fin	Nombre_Senador
2	44	46	ValKilmerAI
3	19	34	MorganFreemanAI
4	0	555	DonaldTrump
5			
6			

Tabla 12: Hablantes que se procesan en la prueba 46

procesamiento_V0020204.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios

```
+ Código + Texto
Q 3/3 [=====] - 6s 1s/step - loss: 0.0829 - accuracy: 0.9730 - val_loss: 0.1182 - val_accuracy: 0.9348
(x) Epoch 15/50
3/3 [=====] - 6s 1s/step - loss: 0.0628 - accuracy: 0.9811 - val_loss: 0.1439 - val_accuracy: 0.9348
Epoch 16/50
3/3 [=====] - 6s 1s/step - loss: 0.0655 - accuracy: 0.9730 - val_loss: 0.1274 - val_accuracy: 0.9565
Epoch 17/50
3/3 [=====] - 6s 2s/step - loss: 0.0423 - accuracy: 0.9838 - val_loss: 0.0978 - val_accuracy: 0.9783
Epoch 18/50
3/3 [=====] - 6s 1s/step - loss: 0.0363 - accuracy: 0.9865 - val_loss: 0.1084 - val_accuracy: 0.9674
Epoch 19/50
3/3 [=====] - 6s 1s/step - loss: 0.0389 - accuracy: 0.9865 - val_loss: 0.0922 - val_accuracy: 0.9783
Epoch 20/50
3/3 [=====] - 6s 1s/step - loss: 0.0250 - accuracy: 0.9892 - val_loss: 0.1373 - val_accuracy: 0.9348
Epoch 21/50
3/3 [=====] - 6s 1s/step - loss: 0.0405 - accuracy: 0.9811 - val_loss: 0.2159 - val_accuracy: 0.9239
Epoch 22/50
3/3 [=====] - 6s 1s/step - loss: 0.0923 - accuracy: 0.9704 - val_loss: 0.0947 - val_accuracy: 0.9674
Epoch 23/50
3/3 [=====] - 6s 1s/step - loss: 0.0412 - accuracy: 0.9811 - val_loss: 0.1179 - val_accuracy: 0.9457
Epoch 24/50
3/3 [=====] - 6s 1s/step - loss: 0.0317 - accuracy: 0.9865 - val_loss: 0.1125 - val_accuracy: 0.9565
Epoch 25/50
3/3 [=====] - 6s 1s/step - loss: 0.0503 - accuracy: 0.9838 - val_loss: 0.1366 - val_accuracy: 0.9457
Epoch 26/50
3/3 [=====] - 6s 1s/step - loss: 0.0379 - accuracy: 0.9919 - val_loss: 0.0893 - val_accuracy: 0.9674
Epoch 27/50
3/3 [=====] - 6s 1s/step - loss: 0.0366 - accuracy: 0.9838 - val_loss: 0.0933 - val_accuracy: 0.9674
Epoch 28/50
3/3 [=====] - 6s 1s/step - loss: 0.0272 - accuracy: 0.9919 - val_loss: 0.1023 - val_accuracy: 0.9674
Epoch 29/50
3/3 [=====] - 6s 1s/step - loss: 0.0362 - accuracy: 0.9838 - val_loss: 0.0959 - val_accuracy: 0.9674
Epoch 30/50
3/3 [=====] - 6s 1s/step - loss: 0.0174 - accuracy: 0.9946 - val_loss: 0.0951 - val_accuracy: 0.9674
Epoch 31/50
3/3 [=====] - 6s 1s/step - loss: 0.0194 - accuracy: 0.9919 - val_loss: 0.0908 - val_accuracy: 0.9783
Epoch 32/50
3/3 [=====] - 6s 1s/step - loss: 0.0155 - accuracy: 0.9946 - val_loss: 0.0945 - val_accuracy: 0.9783
Epoch 33/50
3/3 [=====] - 6s 1s/step - loss: 0.0345 - accuracy: 0.9838 - val_loss: 0.1233 - val_accuracy: 0.9565
Epoch 34/50
3/3 [=====] - 6s 1s/step - loss: 0.0349 - accuracy: 0.9865 - val_loss: 0.1171 - val_accuracy: 0.9674
Epoch 35/50
3/3 [=====] - 6s 1s/step - loss: 0.0884 - accuracy: 0.9757 - val_loss: 0.3302 - val_accuracy: 0.9022
Epoch 36/50
3/3 [=====] - 6s 1s/step - loss: 0.0435 - accuracy: 0.9865 - val_loss: 0.1026 - val_accuracy: 0.9674
```

RAM del sistema
RAM de la GPU
Disco

Gestionar sesiones Cambiar tipo de entorno de ejecución

```
1 print(model.evaluate(valid_ds))
```

3/3 [=====] - 1s 226ms/step - loss: 0.0893 - accuracy: 0.9674
[0.08929096162319183, 0.967391312122345]

Ilustración 127: Métricas para la prueba 46.

```

rnd [6 0 9 2 2 0 1 9 3 6]
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: DonaldTrump Predicted: DonaldTrump
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: DonaldTrump Predicted: DonaldTrump
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: DonaldTrump Predicted: DonaldTrump
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: DonaldTrump Predicted: DonaldTrump
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: DonaldTrump Predicted: DonaldTrump
▶ 0:00/0:01 ━━━━ ⏪ ⏴
Speaker: ValKilmerAI Predicted: ValKilmerAI
▶ 0:00/0:01 ━━━━ ⏪ ⏴

```

Ilustración 128: Resultados para la prueba 46.

```

DonaldTrump
y_pred(6)<=9.557744106867062e-36 JuanManuelMiguez
y_pred(86)<=2.4805065708586635e-37 MarcosCarasso
y_pred(115)<=2.379463534855164e-37 AlejandraVigo
y_pred(116)<=1.15972466939951e-37 RicardoGuerra
y_pred(121)<=1.4130416581700728e-31 MartinLousteau
y_pred(133)<=3.1053387708657815e-20 ValKilmerAI
y_pred(136)<=2.4153601211121534e-13 MorganFreemanAI
y_pred(138)<=1.0 DonaldTrump
1) proba.cercanía=[1.0] orden_categoria=l38 nombre_categoria=DonaldTrump
2) predicción(Voz en Vivo) = DonaldTrump ... y_pred=[l38] index=l39
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [l38]

```

Ilustración 129: Predicción para la prueba 46.

La voz de Donald Trump Deep Fake es reconocida con certeza como la voz de Donald Trump Original.

Prueba 47 – Voz generada por AI

Se agregó la voz de Donald Trump Original.

Se realizó la predicción con la voz de Donald Trumpo sin AI.

The screenshot shows a Jupyter Notebook interface with the following details:

- Toolbar:** Archivo, Editar, Ver, Insertar, Entorno de ejecución, Herramientas, Ayuda, Se han guardado todos los cambios.
- Code Cell:**

```
+ Código + Texto
y_pred(00)<=>3.4900920990012299e-22 MarisolCarasso
y_pred(08)<=>2.630471592633703e-32 EmiliaNyacobitti
y_pred(01)<=>7.799323226717112e-36 VictoriaTolosaPaz
y_pred(02)<=>4.108811601604964e-36 AlejandroRodriguez
y_pred(04)<=>2.5466644697918127e-37 RodrigoDeLoredo
y_pred(05)<=>3.493741773214539e-31 MarioLeogni
y_pred(06)<=>8.881310269897617e-31 LucianoLaspiña
y_pred(08)<=>3.5259402679503703e-32 MaríaBelenTapia
y_pred(09)<=>6.318232200118759e-35 LuciaCorradi
y_pred(101)<=>2.331589403735254e-34 PabloBlanco
y_pred(103)<=>4.279992880101944e-36 AnabelFernandezSagasti
y_pred(105)<=>1.873486945384990e-37 VictorZimmermann
y_pred(106)<=>3.881010810000000e-38 GiovannaOscarPin
y_pred(107)<=>1.75245738451e-33 GuillermoSnoppek
y_pred(108)<=>6.245444905302360e-38 LeonelMujica
y_pred(110)<=>7.84931258180772e-30 MarioF Lad
y_pred(111)<=>1.4328413229630633e-28 GladysGonzalez
y_pred(114)<=>1.346122349086517e-28 MaríaTeresaGonzalez
y_pred(115)<=>4.13562486606365315e-27 RicardoGuerra
y_pred(116)<=>8.31678373562536e-35 CarmenAlvarezRivero
y_pred(121)<=>1.1296546727798493e-21 MartinLousteau
y_pred(122)<=>1.35620977436930053e-32 MauriceCloss
y_pred(126)<=>3.182357770829959e-34 HumbertoSchivoni
y_pred(127)<=>1.2453899502187912e-31 AlfredoCornejo
y_pred(131)<=>3.84631761298889e-32 CristinaFernandezDeKirchner
y_pred(132)<=>1.3993800062772602e-31 CristinaFernandezDeKirchner_hace2años
y_pred(133)<=>1.3993800062772602e-09 ValKilmerAI
y_pred(134)<=>1.159725938864593e-27 ValKilmer
y_pred(136)<=>0.00010721225015569478 MorganFreemanAI
y_pred(138)<=>0.999802768230438e- DonaldTrump
1) proba_cercania=[0.9998027682304382] orden_categoria=138 nombre_categoria=DonaldTrump
2) predicción(Voz en Vivo) = DonaldTrump ... y_pred=[138] index=139
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [138]
```
- Resource Monitor:** RAM, Disco, Recursos X (GPU).

Ilustración 130: Predicción para la prueba 47.

La voz original de Donald Trump fue reconocida como la voz original de Donald Trump (bien)

Prueba 48 – Voz generada por AI

Se agregó la voz de Morgan Freeman con AI.

Se realizó la predicción con la voz de Morgan Freeman con AI.

The screenshot shows a Jupyter Notebook interface with the following details:

- Code Cell:**

```
1) proba_cercania=[0.7325103878974915] orden_categoria=138 nombre_categoria=DonaldTrump
2) predicción(Voz en Vivo) = DonaldTrump ... y_pred=[138] index=139
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [138]
```
- Resource Monitor:** RAM, Disco, Recursos X (GPU).

Ilustración 131: Predicción para la prueba 48.

La voz Deep Fake de Morgan Freeman no se puede predecir.

Prueba 49 – Voz generada por AI

Se agregó la voz de Val Kilmer sin AI.

Se realizó la predicción con la voz de Val Kilmer sin AI.

```

procesamiento_V0020204.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
y_Pred(1)<-3. 5129216523150304e-38 DanyaTavela
y_Pred(51)<-1. 2967643724016627e-32 BlancaOsuna
y_Pred(56)<-2. 430184623181723e-37 AlejandroFinocchiaro
y_Pred(60)<-3. 1076282421524765e-35 GabrielLaRouwerDeKoning
y_Pred(61)<-4. 754925489206826e-38 AnahiCosta
y_Pred(66)<-5. 210192783701822e-38 MariaVictoriaTejeda
y_Pred(70)<-6. 447093458583378e-38 GracielaCafá
y_Pred(71)<-8. 010827778605843e-33 LíaVeronicaCaliva
y_Pred(78)<-4. 225368756117112e-36 CarlosAnibalCisneros
y_Pred(81)<-9. 930666250000000e-38 SantiagoCarlosPolini
y_Pred(85)<-2. 700666250000000e-36 CarolinaMoises
y_Pred(86)<-5. 1.571093993094347e-29 MarianaCarasso
y_Pred(68)<-2. 679580886809792e-38 EmilianoYacobitti
y_Pred(45)<-2. 082399866623928e-37 MarioNicolás
y_Pred(96)<-2. 1.57678923288479e-36 LucianoLaspina
y_Pred(98)<-3. 6.00012631377491e-37 MaríaBelenTapia
y_Pred(110)<-9. 506150404646615e-37 MarioFiad
y_Pred(111)<-<1. 298979025319127e-33 GladysGonzalez
y_Pred(114)<-1. 4724600683820915e-39 MaríaTeresaGonzalez
y_Pred(115)<-5. 466155108866409e-29 AlejandraVigo
y_Pred(116)<-3. 4982335722279013e-32 RicardoGuerra
y_Pred(121)<-2. 163197413787503e-24 MartínLousteau
y_Pred(132)<-2. 2034031761072105e-39 CristinaFernandezDeKirchner_hace22años
y_Pred(133)<-4. 181686604971806e-15 ValKilmerAI
y_Pred(134)<-1. 16686604971806e-15 ValKilmer
y_Pred(136)<-4. 495774138000000e-15 MorganFreemanAI
y_Pred(138)<-0. 999955000000000e-134 DonaldTrump
1) proba.cercania[0.999955000000000e-134] orden_categoria=138 nombre_categoria=DonaldTrump
2) predicción(Voz en Vivo) = DonaldTrump ... y_pred=[138] index=139
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [138]

```

Ilustración 132: Predicción para la prueba 49.

La voz de Val Kilmer original, se confundió con la voz de Donald Trump original.

Prueba 50 – Voz generada por AI

Se agregó la voz de Javier Milei (sin AI)

Se realizó la predicción con la voz de Javier Milei (sin AI)

```

procesamiento_V0020204.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
+ Código + Texto
y_Pred(1)<-3. 202097688012368e-13 MartínLousteau
y_Pred(11)<-3. 335465377240973e-25 MauricioCloss
y_Pred(114)<-8. 39742598207583e-18 MaríaTeresaGonzalez
y_Pred(85)<-2. 56800638492684e-15 AlejandraVigo
y_Pred(116)<-<1. 768712128823264e-16 RicardoGuerra
y_Pred(117)<-1. 709718362527696e-26 PabloYedlin
y_Pred(118)<-1. 43515912085972e-22 GuillermoAñadra
y_Pred(119)<-1. 5166993072409516e-23 CarmenAlvarezRivero
y_Pred(120)<-8. 8212674675393342e-25 IgnacioTorres
y_Pred(121)<-7. 682097688012368e-13 MartínLousteau
y_Pred(122)<-7. 72161598837858e-20 MauriceCloss
y_Pred(123)<-2. 32548795343863e-23 BeatrizVila
y_Pred(124)<-1. 15512561133163e-25 OscarParrilli
y_Pred(125)<-<1. 97381434847567e-24 JuanCarlosRomero
y_Pred(126)<-3. 419982443127804e-28 HumbertoSchivani
y_Pred(127)<-1. 306992454000000e-19 AlfredoCornejo
y_Pred(128)<-1. 310089324371230e-23 MaríaEugenioCatalfamo
y_Pred(129)<-4. 330089324371230e-23 MaríaEugenioCatalfamo
y_Pred(130)<-4. 940768768236781e-27 MaríaLaraVega
y_Pred(131)<-1. 9613755395629900e-22 CristinaFernandezDeKirchner
y_Pred(132)<-6. 152083427559545e-18 CristinaFernandezDeKirchner_hace22años
y_Pred(133)<-<1. 117362686654159e-07 ValKilmerAI
y_Pred(134)<-2. 212627476963876e-16 ValKilmer
y_Pred(135)<-2. 458097959561182e-23 Downloads
y_Pred(136)<-0. 8110645388247288e-25 MorganFreemanAI
y_Pred(137)<-0. 50464232602782e-25 MorganFreeman
y_Pred(138)<-0. 988953136444092 DonaldTrump
y_Pred(139)<-1. 0564880070335457e-25 DonaldTrumpAI
1) proba.cercania[0.988953136444092] orden_categoria=138 nombre_categoria=DonaldTrump
2) predicción(Voz en Vivo) = DonaldTrump ... y_pred=[138] index=139
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [138]

```

Ilustración 133: Predicción para la prueba 50.

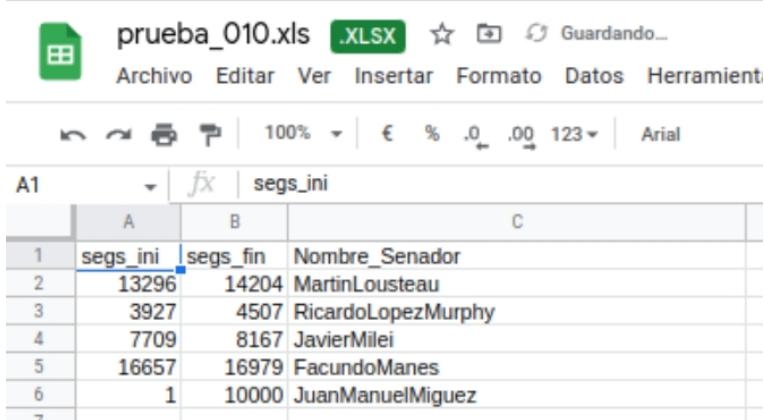
La voz de Javier Milei fue confundida con la voz original de Donald Trump.

Problema: Para que no ocurran falsos positivos, hay que entrenar el modelo de manera pareja con un set de datos que tenga una cantidad balanceada de archivos de audio en cada hablante.

Prueba 51 – Voz sin diafonía, con resfrío

Se agregó la voz de Juan Manuel Miguez sin diafonía y con resfrío:

Se utiliza la voz que en Vivo con resfrío del 16/09/2022



	A1	fx	segs_ini	C
1	segs_ini	segs_fin	Nombre_Senador	
2	13296	14204	MartinLousteau	
3	3927	4507	RicardoLopezMurphy	
4	7709	8167	JavierMilei	
5	16657	16979	FacundoManes	
6	1	10000	JuanManuelMiguez	
7				

Tabla 13: Hablantes que se procesan en la prueba 51

- procesamiento_V0020301.ipynb
- df = prueba_010.xls
- VALID_SPLIT = 0.1 ⇒ VALID_SPLIT = 0.2 y GPU
- EPOCHS = 50

```

procesamiento_V0020301.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
RAM Disco Comentario Compartir Editar Recursos x ...
+ Código + Texto
13/13 [=====] - 2ls ls/step - loss: 0.0933 - accuracy: 0.9619 - val_loss: 0.1590 - val_accuracy: ↑ ↓ ⌂
13/13 [=====] - 2ls ls/step - loss: 0.1067 - accuracy: 0.9651 - val_loss: 0.1331 - val_accuracy: 0.9600
13/13 [=====] - 20s ls/step - loss: 0.0742 - accuracy: 0.9738 - val_loss: 0.2361 - val_accuracy: 0.9200
13/13 [=====] - 20s ls/step - loss: 0.0591 - accuracy: 0.9794 - val_loss: 0.2374 - val_accuracy: 0.9425
13/13 [=====] - 2ls ls/step - loss: 0.0511 - accuracy: 0.9807 - val_loss: 0.1692 - val_accuracy: 0.9625
13/13 [=====] - 2ls ls/step - loss: 0.0774 - accuracy: 0.9669 - val_loss: 0.2421 - val_accuracy: 0.9225
13/13 [=====] - 20s ls/step - loss: 0.0583 - accuracy: 0.9763 - val_loss: 0.1967 - val_accuracy: 0.9400
13/13 [=====] - 20s ls/step - loss: 0.0568 - accuracy: 0.9794 - val_loss: 0.1437 - val_accuracy: 0.9600
13/13 [=====] - 20s ls/step - loss: 0.0600 - accuracy: 0.9744 - val_loss: 0.2019 - val_accuracy: 0.9300
13/13 [=====] - 20s ls/step - loss: 0.0649 - accuracy: 0.9732 - val_loss: 0.1153 - val_accuracy: 0.9650
13/13 [=====] - 20s ls/step - loss: 0.0600 - accuracy: 0.9807 - val_loss: 0.1175 - val_accuracy: 0.9700
13/13 [=====] - 20s ls/step - loss: 0.0606 - accuracy: 0.9763 - val_loss: 0.1365 - val_accuracy: 0.9625
13/13 [=====] - 2ls ls/step - loss: 0.0374 - accuracy: 0.9875 - val_loss: 0.1841 - val_accuracy: 0.9450
13/13 [=====] - 2ls ls/step - loss: 0.0311 - accuracy: 0.9875 - val_loss: 0.2074 - val_accuracy: 0.9400
13/13 [=====] - 20s ls/step - loss: 0.0649 - accuracy: 0.9713 - val_loss: 0.2287 - val_accuracy: 0.9475
13/13 [=====] - 20s ls/step - loss: 0.0451 - accuracy: 0.9819 - val_loss: 0.1171 - val_accuracy: 0.9700
13/13 [=====] - 20s ls/step - loss: 0.0303 - accuracy: 0.9804 - val_loss: 0.1628 - val_accuracy: 0.9675
13/13 [=====] - 20s ls/step - loss: 0.0217 - accuracy: 0.9925 - val_loss: 0.1863 - val_accuracy: 0.9700
13/13 [=====] - 20s ls/step - loss: 0.0129 - accuracy: 0.9963 - val_loss: 0.2414 - val_accuracy: 0.9650
13/13 [=====] - 20s ls/step - loss: 0.0110 - accuracy: 0.9956 - val_loss: 0.2190 - val_accuracy: 0.9700
[ ] 1 print(model.evaluate(valid_ds))
[ ] 1 #redefine BATCH_SIZE, por si la cantidad de muestras es menor a 128
2 BATCH_SIZE = 10
3
25 min 10 s compilado a las 11:37

```

```

1 print(model.evaluate(valid_ds))

13/13 [=====] - 4s 255ms/step - loss: 0.1153 - accuracy: 0.9650
[0.11534533649682999, 0.9649999737739563]

```

Ilustración 134: Predicción para la prueba 51.

```

procesamiento_V0020301.ipynb ☆
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se han guardado todos los cambios
RAM Disco Comentario Compartir Editar Recursos x ...
+ Código + Texto
13/13 [07:49:11.3 0.102 47-30-07 114-22]
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: JavierMiles Predicted: JavierMiles
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: MartinLousteau Predicted: MartinLousteau
Speaker: MartinLousteau Predicted: MartinLousteau
[ ] 1
25 min 10 s compilado a las 11:37

```

Ilustración 135: Resultados para la prueba 51.

```

10 valor_labels += []
11 test.ds = paths_and_labels_to_dataset(valid_audio_paths_vivo, [6])
12 test.ds = test.ds.shuffle(buffer_size=BATCH_SIZE * 8, seed=SHUFFLE_SEED).batch(BATCH_SIZE)
13 test.ds = test.ds.map(lambda x, y: (add_noise(x, noises, scale=SCALE), y))
14 for audios, labels in test.ds.take(1):
15     # Obtener la señal FFT
16     ffts = audio_to_fft(audios)
17     # Predecir
18     y_pred = model.predict(ffts)
19     max_pred_value0
20     idx_max_pred_value0
21     name_max_pred_value0
22     pred_name = class_names[np.argmax(y_pred)]
23     print(pred_name)
24     #print("2) -- {} () ".format(pred_name, y_pred))
25     for index, value in enumerate(y_pred[0]):
26         if [value > 0.0]:
27             print('y_pred[{}]={}<={}'.format(index,value, class_names[index]))
28             if value>max_pred_value0:
29                 max_pred_value0 = value
30                 idx_max_pred_value0 = index
31                 name_max_pred_value0 = pred_name
32             print("1) proba cercania[{}], orden_categoria={}, nombre_categoria={}".format(max_pred_value0, idx_max_pred_value0, name_max_pred_value0))
33             y_pred = np.argmax(y_pred, axis=1)
34             print("2) predicción(Voz en Vivo) = {} ... y_pred[{}]={} index={}".format(class_names[y_pred[0]], y_pred, 0, index))
35             print("3) registro (Voz en Vivo) = {} ({})".format(valid_audio_paths_vivo[0], y_pred, 0))
36
37     display(Audio(valid_audio_paths_vivo[0], rate=SAMPLING_RATE))

JuanManuelMiguez
y_pred[0]<=1.0 orden_categoria=6 nombre_categoria=JuanManuelMiguez
1) proba.cercania=[1.0] orden_categoria=6 nombre_categoria=JuanManuelMiguez
2) predicción(Voz en Vivo) = JuanManuelMiguez ... y_pred=[6] index=139
3) registro (Voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/audio/VozViva/VozViva.wav - [6]

```

Ilustración 136: Predicción para la prueba 51.

La predicción usando la Voz Viva con resfrío, y el modelo entrenado con 5 voces, incluyendo la misma voz, pero sin resfrío, fue de 1 (certeza total)

El modelo predijo que la voz con resfrío es totalmente compatible con el hablante sin resfrío.

Problemas encontrados

No se puede reproducir audio mpeg4 de YouTube

Instalar codec plugin “*GStreamer Multimedia Codecs*” para reproducir archivos de audio mpeg4 descargados desde youtube.

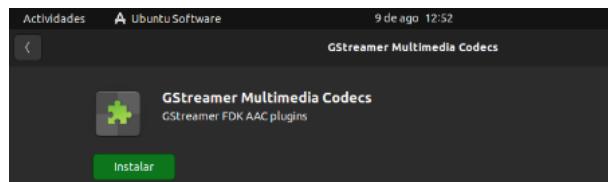


Ilustración 137: Instalación *GStreamer Multimedia Codecs*.

Luego, instalar Audacity como herramienta de reproducción de audio.

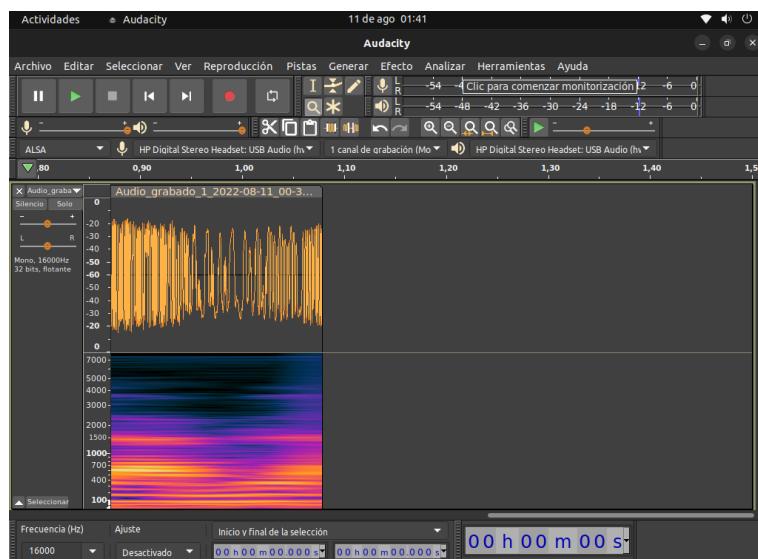


Ilustración 36: Grabación de audio con aplicación Audacity

Error con permisos de ejecución en Anaconda.

El archivo nbserver-20665-open.html no tenía permitido su acceso. El navegador mostraba la leyenda “Acceso denegado al archivo”.



Desde una consola, se generó el archivo de configuración “jupyter_notebook_config.py” en el directorio “`~/jupyter`” con el siguiente comando:

```
$ jupyter notebook --generate-config
```

Luego se editó el mismo para configurar el parámetro “`c.NotebookApp.use_redirect_file`” al valor “`False`”

Finalmente, se otorgaron permisos de ejecución sobre los siguientes directorios: “`~/local/share/jupyter`” “`~/jupyter`” al usuario que ejecuta Anaconda.

Error de dimensionado de pantalla al abrir Anaconda.

“*Can't detect system scaling factor settings for primary monitor.*” Es un problema de configuración.

Se editó el archivo “`anaconda-navigator.ini`”, que está en el directorio “`.anaconda/navigator`”, y se configuró la propiedad “`enable_high_dpi_scaling`” al valor “`False`”

Mensajes de advertencia al abrir Anaconda.

Para resolver los mensajes sobre Links que no funcionan, se debe instalar la última versión de gcc desde un terminal, con los siguientes comandos:

```
$ conda install libgcc
```

Buscar los links que no funcionan en los mensajes de pantalla cuando se desplegó Anaconda:

```
$ sudo find / -wholename "*conda**/libstdc++.so*"
```

Eliminar los links que no funcionan del directorio específico:

```
$ rm /home/jmmiguez/anaconda3/lib/libstdc++*
```

Cambiar en navegador por defecto de Firefox a Chrome

Editar desde línea de comandos el archivo: “`~/jupyter/jupyter_notebook_config.py`” para quitar el comentario de la entrada “`c.NotebookApp.browser = ""`”

Luego, buscar desde dónde se ejecuta Google Chrome en el Sistema Operativo local: en este caso: ‘`/usr/bin/google-chrome`’ y reemplazar en la línea que se está editando: “`c.NotebookApp.browser = '/usr/bin/google-chrome'`”

Escasos recursos de memoria y de disco en Google Colab

Una de las librerías de ploteo, consumió más memoria de lo esperado al graficar las señales de audio en función del tiempo, y también al graficar el espectrograma.

Como resultado, la sesión se quedó sin memoria y el proceso terminó abruptamente, causando la pérdida de información procesada hasta el momento.

Durante la conversión de archivos MP4 a WAV, el espacio en disco se quedó sin espacio. Esto causó una interrupción abrupta, impidiendo continuar con el procesamiento.

Debido a esto, se descartó usar Google Colab hasta resolver ambos problemas. Mientras tanto, se usó Anaconda con Jupyter Notebook en un entorno local. Luego, se eliminaron las librerías de ploteo del desarrollo para ganar performance y continuar utilizando Google Colab.

Límite de ejecuciones en Google Colab

Con el fin de ejecutar más de una notebook al mismo tiempo con procesamiento GPU o TPU, se utilizaron 2 terminales de igual Hardware.

Sin embargo, para limitar el uso de recursos, Google Colab no permite usar más de un tipo de procesador al mismo tiempo para la misma cuenta (1 TPU, 1 GPU) y además, luego de la décima prueba, Google Colab no permitió volver a tomar otra sesión con procesamiento por TPU ni GPU. Esto provocó demoras aproximadas de 4 horas para volver a poner en marcha las pruebas.

Para aprovechar el tiempo, se utilizó el procesamiento genérico para descargar archivos de audio por Deep Fake para terminar las pruebas.

Método predict, entrenando la muestra de un hablante

Se procesaron los archivos de un sólo hablante para entrenamiento y validación. Luego, se ingresó el audio de un hablante diferente y el modelo predijo que el hablante era compatible con certeza absoluta, al hablante que se usó para entrenar el modelo (resultado equivocado)

```
1 history = model.fit(  
2     train_ds,  
3     epochs=100CHS,  
4     validation_data=valid_ds,  
5     callbacks=[earlystopping_cb, mdlcheckpoint_cb],  
6 )  
  
Epoch 1/3  
7/7 [=====] - 151s 21s/step - loss: 0.7611 - accuracy: 0.8513 - val_loss: 0.0000e+00 - va  
L_accuracy: 1.0000  
Epoch 2/3  
7/7 [=====] - 175s 24s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00  
- val_accuracy: 1.0000  
Epoch 3/3  
7/7 [=====] - 167s 23s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00  
- val_accuracy: 1.0000  
  
1 print(model.evaluate(valid_ds))  
3/3 [=====] - 5s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000  
[0.0, 1.0]
```

Ilustración 37: Entrenamiento y evaluación, con 1 categoría.

Para descartar malas prácticas que conducen a este tipo de resultados (como sobre-entrenamiento), se volvió a ejecutar cada conjunto de prueba con diferentes valores de VALID_SPLIT (desde 0,1 hasta 0,9 con intervalos incrementales de +0,1). Además, se modificó la cantidad de archivos de muestra, entre 50, 100, 200 y 400 archivos. Finalmente, todas las pruebas devolvieron el mismo resultado.

El método `model.predict` devuelve un array[categorías] de probabilidades de cercanía a cada categoría. Si el modelo entrena con 1 categoría, entonces la certeza será total con cualquier dato nuevo que se intente predecir. Si el modelo entrena con más de 1 categoría, entonces la probabilidad puede tender a 1 (uno) en el caso de certeza, para manifestar compatibilidad, o a 0 (cero) en caso contrario.

Código fuente

Preprocesamiento

https://colab.research.google.com/drive/1ilSD1qZ-gA_D_qQzAlyh9oX42iIEzR2A#scrollTo=d506314b

Notebook para pre-procesar archivos de voz v0030101

Instalar librerías, paquetes y dependencias

```
from google.colab import drive
drive.mount('/content/drive/')

!pip install pytube

!apt -qq install -y sox
!apt -qq install -y sox libsox-fmt-mp3
!pip install sox

!git clone https://github.com/rabitt/pysox.git
!cd /content/pysox
!python /content/pysox/setup.py install

!pip install git+https://github.com/rabitt/pysox.git

!pip3 install ffmpeg
!apt -qq install -y ffmpeg
!pip3 install pydub

!pip3 install xlrd
!pip3 install --upgrade xlrd

!pip3 install pandas
!pip3 install --upgrade pandas

Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from sox==1.4.2) (1.21.6)
Requirement already satisfied: typing-extensions>=3.7.4.2 in /usr/local/lib/python3.7/dist-packages (from sox==1.4.2) (4.1.1)
Building wheels for collected packages: sox
  Building wheel for sox: setup.py ... done
    Created wheel for sox: filename=sox-1.4.2-py2.py3-none-any.whl size=40190 sha256=0defee7977af081e69cf026225b7959f2563e86eb2069d8e49a4f5fa0127667
    Stored in directory: /tmp/pip-ephem-wheel-cache-gkrl2zhk/wheels/8f/6c/f9/e71605a7cc6e5905903c60458159bdac02024650b3dae5feac
Successfully built sox
Installing collected packages: sox
  Attempting uninstall: sox
    Found existing installation: sox 1.4.1
    Uninstalling sox-1.4.1:
      Successfully uninstalled sox-1.4.1
Successfully installed sox-1.4.2
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ffmpeg
  Downloading ffmpeg_1.4.tar.gz (5.1 kB)
...
Installing collected packages: xlrd
  Attempting uninstall: xlrd
    Found existing installation: xlrd 1.1.0
    Uninstalling xlrd-1.1.0:
      Successfully uninstalled xlrd-1.1.0
Successfully installed xlrd-2.0.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2022.2.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2022.2.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
```

Ilustración 138: Instalar librerías, paquetes y dependencias.

Importar librerías, paquetes y dependencias

```
import os
import subprocess
import librosa
import librosa.display
import sox
import pandas
import string
import traceback
import sys
from pytube import YouTube
from pathlib import Path
from IPython.display import Audio
from pydub import AudioSegment
from pydub.utils import make_chunks
```

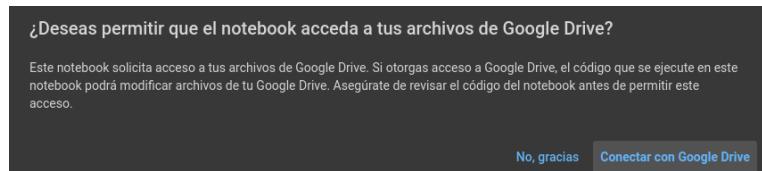
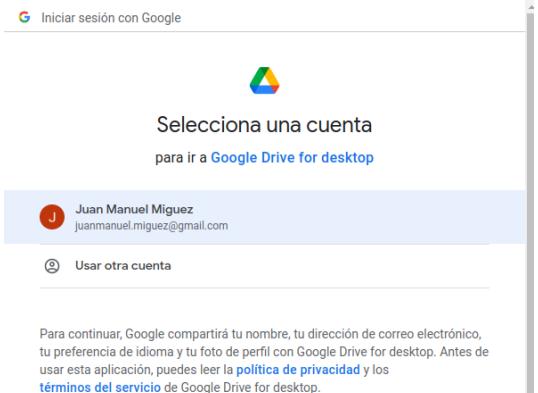
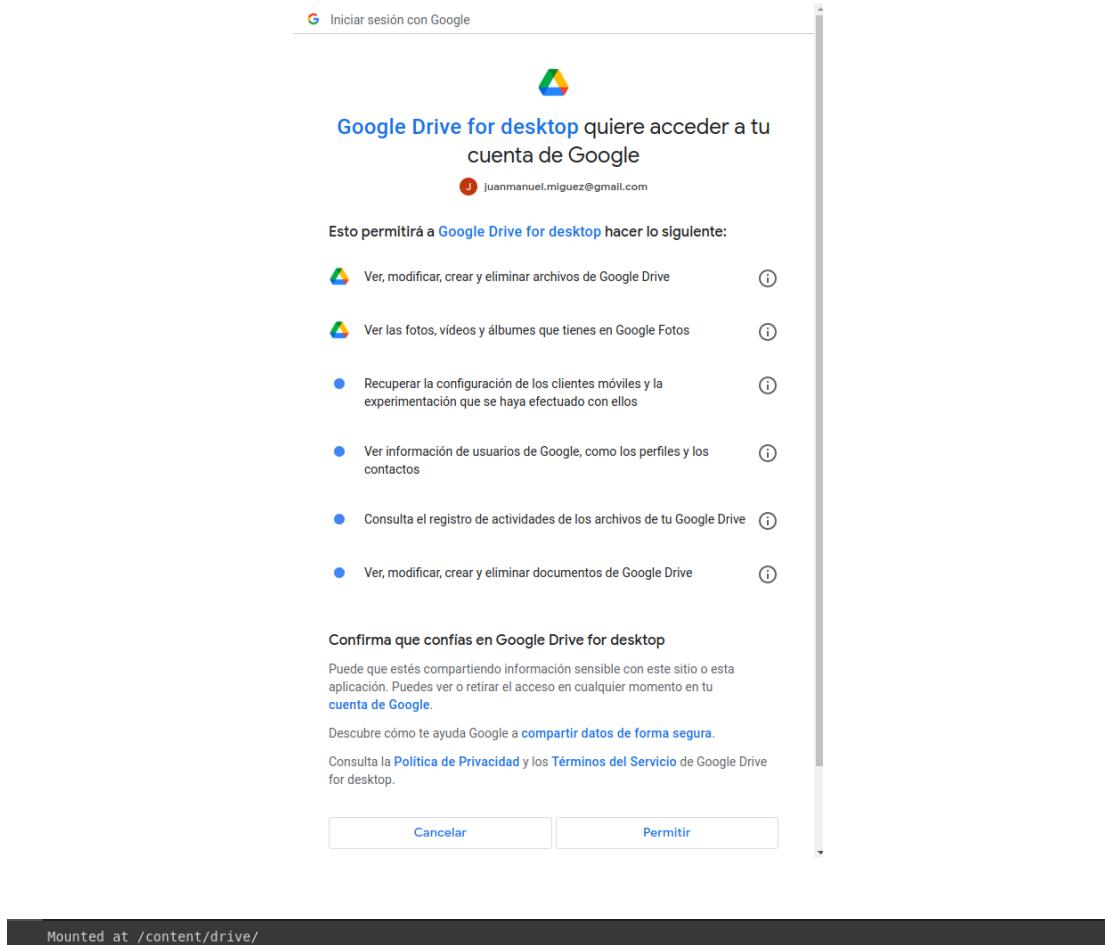


Ilustración 139: Conectar con Google Drive





Constantes y Variables de sesión

```
# Constantes y Variables de sesión

#C_MAIN_AUDIO_PATH = '/home/jmmiguez/proyectoAudio/audio/'
C_MAIN_AUDIO_PATH = '/content/drive/MyDrive/proyectoAudio/audio/'

#C_DATASET_AUDIO_PATH = "/home/jmmiguez/proyectoAudio/audio/Downloads/"
C_DATASET_AUDIO_PATH = "/content/drive/MyDrive/proyectoAudio/audio/Downloads/"

V_NOMBRE_ARCHIVO_MAESTRO = 'audio_maestro'
V_PATH_Y_NOMBRE_ARCHIVO_MP4 = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.mp4'
V_PATH_Y_NOMBRE_ARCHIVO_WAV = C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'

#dirección del video de descarga en YouTube
V_VIDEO_LINK = 'https://youtu.be/aPp5lcqgISk

```

Funciones para descargar audio y convertir a *.wav

```
# Funciones para descargar audio y convertirlo a WAV

def cargarAudio():
    print('2.0. Cargar audio')
    try:
        if os.path.isfile(C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'):
            V_COMMAND = "rm " + C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav'
            print('2.1. command >>', V_COMMAND)
            subprocess.call(V_COMMAND, shell=True)
            print('2.2. Archivo mp4 eliminado')

        if not(os.path.exists(C_DATASET_AUDIO_PATH + V_NOMBRE_ARCHIVO_MAESTRO + '.wav')):
            V_VIDEO = YouTube(V_VIDEO_LINK)
            V_AUDIO = V_VIDEO.streams.filter(only_audio=True).first()
            if V_AUDIO.subtype == 'mp4':
                V_AUDIO.download(C_DATASET_AUDIO_PATH,
filename=V_PATH_Y_NOMBRE_ARCHIVO_MP4, filename_prefix='')
                print('2.3. Descargó el archivo: "{}{}".'.format(C_DATASET_AUDIO_PATH,
V_NOMBRE_ARCHIVO_MAESTRO+'.'+V_AUDIO.subtype))
            except:
                print("Error: cargarAudio")
                traceback.print_exc()
    return None

def convertirMP4aWAV():
    print('3.0. Convertir el archivo mp4 a wav')
    try:
        if not(os.path.exists(V_PATH_Y_NOMBRE_ARCHIVO_WAV)):
            V_COMMAND = "ffmpeg -i " + " " + V_PATH_Y_NOMBRE_ARCHIVO_MP4 + " " + "-ar
16000 -vn" + " " + V_PATH_Y_NOMBRE_ARCHIVO_WAV
            print('3.1. $ ', V_COMMAND)
            subprocess.call(V_COMMAND, shell=True)
            print('3.2. archivo convertido de mp4 a wav')
        except:
            print("Error: convertirMP4aWAV")
            traceback.print_exc()
    return None

def identificarArchivoAudioSinPreProcesar():
    print('4.0. Identificar los archivos de audio sin pre-procesar')
    try:
        V_AUDIO_PATHS = []
        for subdir in os.listdir(C_DATASET_AUDIO_PATH):
            V_SUBDIR_PATH = Path(C_DATASET_AUDIO_PATH) / subdir
            V_LAST_PART = V_SUBDIR_PATH.parts[-1]
            if V_LAST_PART.endswith(".wav"):
                print('4.1. {}'.format(V_SUBDIR_PATH))
                V_AUDIO_PATHS += [V_SUBDIR_PATH]
        print("4.2. Encontró {} archivo/s, en {}".format(len(V_AUDIO_PATHS), len(os.listdir(C_DATASET_AUDIO_PATH))))
        for V_INDEX in (V_AUDIO_PATHS):
            print('4.3. Archivo encontrado V_INDEX = {}'.format(V_INDEX))

    except:
        print("Error: identificarArchivoAudioSinPreProcesar")
        traceback.print_exc()
    return V_INDEX

def eliminarArchivoMP4():
    try:
        if os.path.isfile(V_PATH_Y_NOMBRE_ARCHIVO_MP4) is True:
            command = ("rm " + "{}".format(V_PATH_Y_NOMBRE_ARCHIVO_MP4))
            print('5.0. command >>', command)
```

```

        subprocess.call(command, shell=True)
        print('5.1. Archivo mp4 eliminado')
    except:
        print("Error: eliminarArchivoMP4")
        traceback.print_exc()
    return None

```

Descargar audio y convertirlo a *.wav

```

#Descargar audio y convertirlo a *.wav
try:
    cargarAudio()
    convertirMP4aWAV()
    V_INDEX = identificarArchivoAudioSinPreProcesar()
    eliminarArchivoMP4()
    print('5.2. Fin correcto - Parte 1')
except:
    print("Error: Nombre del hablante")
    traceback.print_exc()

```

```

2.0. Cargar audio
2.3. Descargó el archivo: "/content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.mp4"
3.0. Convertir el archivo mp4 a wav
3.1. $ ffmpeg -i /content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.mp4 -ar 16000 -vn /content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.wav
3.2. Eliminó el archivo de mp4
4.0. Identificar los archivos de audio sin pre-procesar
4.1. /content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.wav
4.2. Encontró 1 archivo/s, en 2 directorios
4.3. Archivo encontrado V_INDEX = /content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.wav
5.0. command >> rm /content/drive/MyDrive/proyectoAudio/audio/Downloads/audio_maestro.mp4
5.1. Archivo mp4 eliminado
5.2. Fin correcto - Parte 1

```

Ilustración 140: Descargar audio y convertirlo a *.wav

Nombre del hablante, Instante de inicio y fin del audio, desde xls

```

df = pandas.read_excel(V_PATH_Y_NOMBRE_ARCHIVO_XLS)
count = df.shape[0]

for index, row in df.iterrows():
    print("segs_ini={} segs_fin={} nombre={}".format(row[0], row[1], row[2]))
    if index == count - 1:
        break

```

```
segs_ini=0.0 segs_fin=26.0 nombre=DonaldTrumpAI
```

Ilustración 141: Nombre del hablante, Instante de inicio y fin del audio, desde xls.

Funciones para normalizar Audio

```

#Funciones para normalizar Audio

def normalizarAudioCrudo(V_SEGS_DESDE, V_SEGS_HASTA):
    try:
        print('7.0. Normalizar el archivo crudo, usando SOX')
        V_TFM = sox.Transformer()
        V_TFM.rate(samplerate=16000)
        V_TFM.norm(db_level=-3)
        print('7.1. Recortar el audio maestro entre segundos de inicio y fin')
        V_TFM.trim(V_SEGS_DESDE, V_SEGS_HASTA)
        V_TFM.silence(
            location = 0,
            silence_threshold = 0.5,
            min_silence_duration = 0.1,
            buffer_around_silence = False
        )
        print('7.2. Definir el nombre del archivo de salida normalizado')
        V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV = "{}"
        "{}_normal.wav".format(C_DATASET_AUDIO_PATH, V_NOMBRE_ARCHIVO_MAESTRO)
    
```

```

        V_TFM.build("{}".format(V_INDEX),
"{}".format(V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV))
        V_TFM.effects_log
    except:
        print("Error: normalizarAudioCrudo")
        traceback.print_exc()
    return V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV

def dividirArchivoNormalizadoEnChunks1seg(V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV,
V_NOMBRE_HABLANTE):
    try:
        print('8.0. Dividir el archivo normalizado, en chunks de 1 segundo.')
        myaudio =
AudioSegment.from_file("{}".format(V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV) , "wav")
        chunks = make_chunks(myaudio, V_CHUNK_LENGTH)

        for i, chunk in enumerate(chunks):
            V_PATH_Y_NOMBRE_ARCHIVO_WAV = C_DATASET_AUDIO_PATH + V_NOMBRE_HABLANTE +
'.wav'
            chunk_name = "{}".format(C_DATASET_AUDIO_PATH) + "{0}.wav".format(i+0)
            #print ("exporting {}".format(chunk_name))
            chunk.export(chunk_name, format="wav")

        command = ("rm " + "{}".format(V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV))
        #print('command >>',command)
        subprocess.call(command, shell=True)

        V_PATH_CHUNK_NORMALIZADO_WAV = C_MAIN_AUDIO_PATH + 'audio/' +
V_NOMBRE_HABLANTE
        try:
            if not(os.path.exists(V_PATH_CHUNK_NORMALIZADO_WAV)):
                #print('command >> mkdir',V_PATH_CHUNK_NORMALIZADO_WAV)
                os.makedirs(V_PATH_CHUNK_NORMALIZADO_WAV)
        except:
            print("Error - al intentar crear directorio:
{}".format(V_PATH_CHUNK_NORMALIZADO_WAV))
            traceback.print_exc()
        try:
            command = ("mv " + "{}".format(C_DATASET_AUDIO_PATH) + "[0-9]*.wav " +
V_PATH_CHUNK_NORMALIZADO_WAV)
            print('8.1. command >>',command)
            subprocess.call(command, shell=True)
        except:
            print("Error - al mover chunk {}".format((C_DATASET_AUDIO_PATH) + "[0-
9]*.wav " + V_PATH_CHUNK_NORMALIZADO_WAV))
            traceback.print_exc()
        except:
            print("Error: dividirArchivoNormalizadoEnChunks1seg")
            traceback.print_exc()
    return None

def eliminarChunks(V_PATH):
    try:
        os.chdir(V_PATH)
        print("removiendo chunks de {}",format(V_PATH))
        for f in os.listdir('.'):
            if any(x in f for x in string.digits) and f.endswith('.wav'):
                try:
                    os.remove(f)
                except:
                    print("Error - al eliminar chunks")
        print("chunks removidos")
    except:
        print("Error - al eliminar chunks {}".format((V_PATH) + "[0-9]*.wav "))
    return None

```

Normalización y división en múltiples muestras

Normalización y división en múltiples muestras

```

df = pandas.read_excel(V_PATH_Y_NOMBRE_ARCHIVO_XLS)
count = df.shape[0]

for index, row in df.iterrows():
    print("segs_ini={} segs_fin={} nombre={}".format(row[0], row[1], row[2]))
    V_SEGS_DESDE=row[0]
    V_SEGS_HASTA=row[1]
    V_NOMBRE_HABLANTE=row[2]
    V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV = normalizarAudioCrudo(V_SEGS_DESDE,
V_SEGS_HASTA)
    eliminarChunks(C_MAIN_AUDIO_PATH + 'audio/' + V_NOMBRE_HABLANTE)
    dividirArchivoNormalizadoEnChunks1seg(V_PATH_Y_NOMBRE_ARCHIVO_NORMALIZADO_WAV,
V_NOMBRE_HABLANTE)
    if index == count - 1:
        break
eliminarChunks(C_DATASET_AUDIO_PATH)
print('Fin correcto - Parte 2')

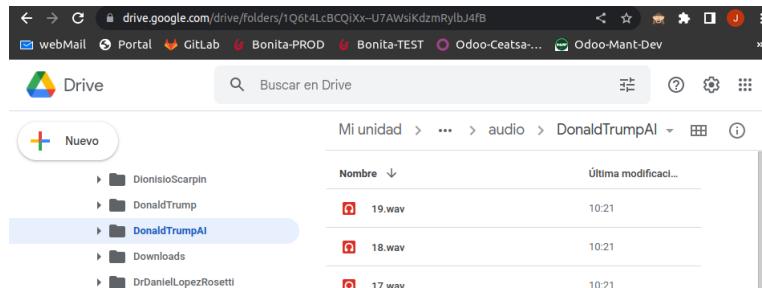
```

```

segs_ini=0.0 segs_fin=26.0 nombre=DonaldTrumpAI
7.0. Normalizar el archivo crudo, usando SOX
7.1. Recortar el audio maestro entre segundos de inicio y fin
7.2. Definir el nombre del archivo de salida normalizado
removiendo chunks de {} /content/drive/MyDrive/proyectoAudio/audio/DonaldTrumpAI
chunks removidos
8.0. Dividir el archivo normalizado, en chunks de 1 segundo.
8.1. command >> mv /content/drive/MyDrive/proyectoAudio/audio/Downloads/*.*.wav /content/drive/MyDrive/proyectoAudio/audio/DonaldTrumpAI
removiendo chunks de {} /content/drive/MyDrive/proyectoAudio/audio/Downloads/
chunks removidos
Fin correcto - Parte 2

```

Ilustración 142: Normalización y división en múltiples muestras.



Procesamiento

<https://colab.research.google.com/drive/1VSzJ07gYwx3lT57zcpaD2Kfn8iwAyk5J#scrollTo=a31a320e>

Notebook para procesar archivos de voz v0070101

Instalar librerías, paquetes y dependencias

#Para descargar videos de Youtube

```

!pip install pytube
#Para procesamiento de audio, convertir formatos de archivos de audio en otros
formatos, convertir tasas de muestreo, para aplicar efectos de sonido, reproducir y
grabar archivos de audio.
!apt -qq install -y sox
!apt -qq install -y sox libsox-fmt-mp3
!pip install sox
!git clone https://github.com/rabitt/pysox.git
!cd /content/pysox

```

```

!python /content/pysox/setup.py install
!pip install git+https://github.com/rabitt/pysox.git

#convertidor de audio y video. Se usa para convertir WAV a MP3 y viceversa.
!pip3 install ffmpeg
!apt -qq install -y ffmpeg

#Para reproducir, dividir, integrar o editar los archivos de audio únicamente con
extensión .wav
!pip3 install pydub

#Para extraer datos de una hoja de cálculo, solo para archivos de extensión .xls
!pip3 install xlrd
!pip3 install --upgrade xlrd
!pip install xlrd
!pip3 install xlrd
!pip install --upgrade --force-reinstall xlrd

#Para manejar y analizar estructuras de datos.
!pip3 install pandas
!pip3 install --upgrade pandas

C+ Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pytube
  Downloading pytube-12.1.0-py3-none-any.whl (56 kB)
    ████████████████████████████████████████████ | 56 kB 3.4 MB/s
Installing collected packages: pytube
Successfully installed pytube-12.1.0
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following additional packages will be installed:

Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2022.6)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)

```

Ilustración 144: Instalar librerías, paquetes y dependencias.

Importar librerías, paquetes y dependencias

```

import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import shutil
import tensorflow as tf
import tensorflow_hub as hub

from IPython.display import display, Audio
from pathlib import Path
from scipy import stats
from scipy.io import wavfile
from sklearn.metrics import confusion_matrix
from tensorflow import keras
from google.colab import drive
drive.mount('/content/drive/')

```

```
C+ Mounted at /content/drive/
```

Ilustración 143: montar Drive.

Variables de sesión

```
# Variables de sesión
```

```

#Usar el path según procesamiento local o cloud.
#DATASET_ROOT = "/home/jmmiguez/proyectoAudio/audio/"
DATASET_ROOT = '/content/drive/MyDrive/proyectoAudio/audio'

# Las carpetas en las cuales voy a poner los ejemplos de audio y los ejemplos de ruidos
AUDIO_SUBFOLDER = "audio"
NOISE_SUBFOLDER = "noise"
DATASET_AUDIO_PATH = os.path.join(DATASET_ROOT, AUDIO_SUBFOLDER)
DATASET_NOISE_PATH = os.path.join(DATASET_ROOT, NOISE_SUBFOLDER)

print ("DATASET_ROOT {}".format(DATASET_ROOT))
print ("DATASET_AUDIO_PATH {}".format(DATASET_AUDIO_PATH))
print ("DATASET_NOISE_PATH {}".format(DATASET_NOISE_PATH))

# Porcentaje de muestras que voy a usar para validación
VALID_SPLIT = 0.2

# Semilla que voy a usar para mezclar los datos con el ruido
SHUFFLE_SEED = 43

# La tasa de muestreo a usar es única para todas las muestras de audio.
# Se vuelve a muestrear todo el ruido a esta frecuencia de muestreo.
# Este también será el tamaño de salida de las muestras de las señales de audio.
# (ya que todas las muestras son de 1 segundo de duración)
SAMPLING_RATE = 16000

# El factor a multiplicar por el ruido, es acorde a:
#   muestra_de_ruido = muestreo + ruido * prop * escala
#   donde prop = amplitud_de_muestreo / amplitud_de_ruido
SCALE = 0.5

BATCH_SIZE = 150
#EPOCHS = 100
EPOCHS = 50

DATASET_ROOT /content/drive/MyDrive/proyectoAudio/audio
DATASET_AUDIO_PATH /content/drive/MyDrive/proyectoAudio/audio/audio
DATASET_NOISE_PATH /content/drive/MyDrive/proyectoAudio/audio/noise

```

Ilustración 145: Variables de sesión.

Administrar estructuras de directorios para archivos

```

# Si la carpeta 'audio' no existe, la creo; caso contrario, no hago nada.
if os.path.exists(DATASET_AUDIO_PATH) is False:
    os.makedirs(DATASET_AUDIO_PATH)

# si la carpeta 'noise' no existe, la creo; caso contrario, no hago nada.
if os.path.exists(DATASET_NOISE_PATH) is False:
    os.makedirs(DATASET_NOISE_PATH)

for folder in os.listdir(DATASET_ROOT):
    if os.path.isdir(os.path.join(DATASET_ROOT, folder)):

        print ("folder = {}".format(folder))

        if folder in [AUDIO_SUBFOLDER, NOISE_SUBFOLDER]:
            # Si la carpeta es 'audio' o 'noise', no hago nada
            continue
        elif folder in ["other", "_background_noise_"]:
            # Si la sub-carpeta es una de las que contienen muestras de ruido,
            # moverla a la carpeta 'noise'
            shutil.move(
                os.path.join(DATASET_ROOT, folder),
                os.path.join(DATASET_NOISE_PATH, folder),
            )
        else:
            # De otra forma, debe ser una carpeta de un hablante, así que hay que
            # moverla a la carpeta de 'audio'
            shutil.move(

```

```

        os.path.join(DATASET_ROOT, folder),
        os.path.join(DATASET_AUDIO_PATH, folder),
    )

```

```

folder = audio
folder = noise

```

Ilustración 146: directorios para archivos.

Archivos de audio con ruido

```

# Obtengo la lista de todos los archivos de ruido
noise_paths = []
for subdir in os.listdir(DATASET_NOISE_PATH):
    subdir_path = Path(DATASET_NOISE_PATH) / subdir

    print("subdir_path= {}".format(subdir_path))

    if os.path.isdir(subdir_path):
        noise_paths += [
            os.path.join(subdir_path, filepath)
            for filepath in os.listdir(subdir_path)
            if filepath.endswith(".wav")
        ]

print(
    "En el directorio {}, se encontró {} archivos, pertenecientes a {}"
    "directorios".format(
        DATASET_NOISE_PATH, len(noise_paths), len(os.listdir(DATASET_NOISE_PATH))
    )
)

print(noise_paths)

```

```

▷ subdir_path= /content/drive/MyDrive/proyectoAudio/audio/noise/audio
subdir_path= /content/drive/MyDrive/proyectoAudio/audio/noise/noise
En el directorio /content/drive/MyDrive/proyectoAudio/audio/noise, se encontró 3 archivos, pertenecientes a 2 directorios
['/content/drive/MyDrive/proyectoAudio/audio/noise/audio/bicicleta.wav', '/content/drive/MyDrive/proyectoAudio/audio/noise/noise/aplausos.wav',

```

Ilustración 147: Archivos de audio con ruido.

Muestreo y división por tiempo de archivos de audio con ruido

```

command = (
    "for dir in `ls -1 " + DATASET_NOISE_PATH + "`; do "
    "for file in `ls -1 " + DATASET_NOISE_PATH + "/$dir/*.wav`; do "
    "sample_rate=`ffprobe -hide_banner -loglevel panic -show_streams "
    "$file | grep sample_rate | cut -f2 -d=`;
    "if [ $sample_rate -ne 16000 ]; then "
    "ffmpeg -hide_banner -loglevel panic -y "
    "-i $file -ar 16000 temp.wav; "
    "mv temp.wav $file; "
    "fi; done; done"
)
os.system(command)

# Dividir el ruido en fragmentos de 16000 Hz cada uno
def load_noise_sample(path):

    print(path)

    sample, sampling_rate = tf.audio.decode_wav(
        tf.io.read_file(path), desired_channels=1
    )
    if sampling_rate == SAMPLING_RATE:
        # Número de cortes a 16000 cada uno, que se pueden generar a partir de la
        muestra de ruido
        slices = int(sample.shape[0] / SAMPLING_RATE)
        sample = tf.split(sample[: slices * SAMPLING_RATE], slices)

```

```

        return sample
    else:
        print("La tasa de muestreo para {} es incorrecta. Se ignora".format(path))
        return None

noises = []
for path in noise_paths:
    sample = load_noise_sample(path)
    if sample:
        noises.extend(sample)
noises = tf.stack(noises)

print( "format(len(noise_paths)=      {}".format(len(noise_paths)) )
print( "noises.shape[0]=            {}".format(noises.shape[0]) )
print( "noises.shape[1]=            {}".format(noises.shape[1]) )
print( "noises.shape[1] // SAMPLING_RATE = {}".format(noises.shape[1] // SAMPLING_RATE) )

print(
    "{} archivos de ruido fueron divididos en {} muestras de ruido, donde cada una tiene {} seg. de tiempo.".format(
        len(noise_paths), noises.shape[0], noises.shape[1] // SAMPLING_RATE)
)

```

```

/content/drive/MyDrive/proyectoAudio/audio/noise/audio/bicicleta.wav
/content/drive/MyDrive/proyectoAudio/audio/noise/noise/aplausos.wav
/content/drive/MyDrive/proyectoAudio/audio/noise/noise/ruido.wav
format(len(noise_paths)=      3
noises.shape[0]=            137
noises.shape[1]=            16000
noises.shape[1] // SAMPLING RATE = 1
3 archivos de ruido fueron divididos en 137 muestras de ruido, donde cada una tiene 1 seg. de tiempo.

```

Ilustración 148: Muestreo y división por tiempo de archivos de audio con ruido.

Funciones para procesar audio

```

# Funciones para procesar Audio

def paths_and_labels_to_dataset(audio_paths, labels):
    """Construir un dataset de audios y etiquetas."""
    path_ds = tf.data.Dataset.from_tensor_slices(audio_paths)
    audio_ds = path_ds.map(lambda x: path_to_audio(x))
    label_ds = tf.data.Dataset.from_tensor_slices(labels)
    return tf.data.Dataset.zip((audio_ds, label_ds))

def path_to_audio(path):
    """Leer y decodificar un archivo de audio."""
    audio = tf.io.read_file(path)
    audio, _ = tf.audio.decode_wav(audio, 1, SAMPLING_RATE)
    return audio

def add_noise(audio, noises=None, scale=0.5):
    if noises is not None:
        # Crear un tensor aleatorio del mismo tamaño que el audio que va
        # desde 0 hasta la cantidad de muestras de flujo de ruido que tenemos.
        tf_rnd = tf.random.uniform(
            (tf.shape(audio)[0],), 0, noises.shape[0], dtype=tf.int32
        )
        noise = tf.gather(noises, tf_rnd, axis=0)

        # Obtener la proporción de amplitud entre el audio y el ruido.
        prop = tf.math.reduce_max(audio, axis=1) / tf.math.reduce_max(noise, axis=1)
        prop = tf.repeat(tf.expand_dims(prop, axis=1), tf.shape(audio)[1], axis=1)

        # Agregar el ruido reescalado al audio
        audio = audio + noise * prop * scale

    return audio

```

```

def audio_to_fft(audio):
    # Dado que tf.signal.fft aplica FFT en la dimensión más interna,
    # debemos comprimir las dimensiones y luego expandirlas nuevamente
    # después de FFT
    audio = tf.squeeze(audio, axis=-1)
    fft = tf.signal.fft(
        tf.cast(tf.complex(real=audio, imag=tf.zeros_like(audio)), tf.complex64)
    )
    fft = tf.expand_dims(fft, axis=-1)

    # Devuelve el valor absoluto de la primera mitad de la FFT
    # que representa las frecuencias positivas
    return tf.math.abs(fft[:, : (audio.shape[1] // 2), :])

```

Obtener las clases y sus etiquetas

```

# Obtener las clases y sus etiquetas

class_names = os.listdir(DATASET_AUDIO_PATH)
print("Posibles nombres de nuestras Clases: {}".format(class_names,))

audio_paths = []
labels = []
class_names_process = []

#Reemplazo la recorrida original de archivos de audio, por los nombres de hablantes
que están en un excel.
#df =
pd.read_excel("/home/jmmiguez/proyectoAudio/audio/DiputadosDebatenElAcuerdoConElFMI3.xls")
#df = pd.read_excel("/home/jmmiguez/proyectoAudio/audio/prueba_004.xls")
df = pd.read_excel("/content/drive/MyDrive/proyectoAudio/audio/prueba_011.xls")

count = df.shape[0]

posicion = 0

for label, name in enumerate(class_names):
    # print("Name={} - Label={}".format(name, label,))

    for index, row in df.iterrows():
        #print("==>son iguales nombre={} y nombre={} +----"
index{}".format(row[2],name,index))

        if (row[2]==name):
            print("Procesando al hablante {}".format(row[2],))
            dir_path = Path(DATASET_AUDIO_PATH) / row[2]
            speaker_sample_paths = [
                os.path.join(dir_path, filepath)
                for filepath in os.listdir(dir_path)
                if filepath.endswith(".wav")]
            ]
            audio_paths += speaker_sample_paths
            labels += [posicion] * len(speaker_sample_paths)
            posicion = posicion + 1
            class_names_process += [name]

        if index == count - 1:
            break

print("Nombres de nuestras Clases (class_names_process):"
    "{}".format(class_names_process,))

print(
    "#Encontrados {} archivos, pertenecientes a {} clases. \
nlabels={}".format(len(audio_paths), len(class_names), labels)
    "Encontrados {} archivos, pertenecientes a {} clases. \nlabels={}\ \
naudio_paths={}".format(len(audio_paths), len(class_names_process), labels,
    audio_paths)
)

```



```
Usando 2357 archivos para entrenamiento.  
Usando 589 archivos para validación.
```

Ilustración 150: Set de datos para entrenamiento y para validación.

Construir el modelo

```
# Construir el modelo: layers, activación, optimizador, loss, métricas, callback con  
early_stopping y checkpoint

def residual_block(x, filters, conv_num=3, activation="relu"):  
    # Atajo  
    s = keras.layers.Conv1D(filters, 1, padding="same")(x)  
    for i in range(conv_num - 1):  
        x = keras.layers.Conv1D(filters, 3, padding="same")(x)  
        x = keras.layers.Activation(activation)(x)  
    x = keras.layers.Conv1D(filters, 3, padding="same")(x)  
    x = keras.layers.Add()([x, s])  
    x = keras.layers.Activation(activation)(x)  
    return keras.layers.MaxPool1D(pool_size=2, strides=2)(x)

def build_model(input_shape, num_classes):  
    inputs = keras.layers.Input(shape=input_shape, name="input")  
  
    x = residual_block(inputs, 16, 2)  
    x = residual_block(x, 32, 2)  
    x = residual_block(x, 64, 3)  
    x = residual_block(x, 128, 3)  
    x = residual_block(x, 128, 3)  
  
    x = keras.layers.AveragePooling1D(pool_size=3, strides=3)(x)  
    x = keras.layers.Flatten()(x)  
    x = keras.layers.Dense(256, activation="relu")(x)  
    x = keras.layers.Dense(128, activation="relu")(x)  
  
    outputs = keras.layers.Dense(num_classes, activation="softmax", name="output")(x)  
  
    return keras.models.Model(inputs=inputs, outputs=outputs)

#model = build_model((SAMPLING_RATE // 2, 1), len(class_names))  
model = build_model((SAMPLING_RATE // 2, 1), len(class_names_process))  
  
model.summary()  
  
#import keras_metrics as km  
#from keras import metrics  
  
model.compile(  
    optimizer="Adam",  
    loss="sparse_categorical_crossentropy",  
    metrics=["accuracy"]  
)  
  
# Agregar devoluciones de llamada:  
# 'EarlyStopping' para dejar de entrenar cuando el modelo ya no mejora.  
# 'ModelCheckPoint' mantener siempre el modelo que tiene el mejor val_accuracy  
model_save_filename = "model.h5"  
  
earlystopping_cb = keras.callbacks.EarlyStopping(patience=10,  
restore_best_weights=True)  
mdlcheckpoint_cb = keras.callbacks.ModelCheckpoint(model_save_filename,  
monitor="val_accuracy", save_best_only=True)
```

Model: "model"			
Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 8000, 1)	0	[]
conv1d_1 (Conv1D)	(None, 8000, 16)	64	['input[0][0]']
activation (Activation)	(None, 8000, 16)	0	['conv1d_1[0][0]']
conv1d_2 (Conv1D)	(None, 8000, 16)	784	['activation[0][0]']
conv1d (Conv1D)	(None, 8000, 16)	32	['input[0][0]']
add (Add)	(None, 8000, 16)	0	['conv1d_2[0][0]', 'conv1d[0][0]']
activation_1 (Activation)	(None, 8000, 16)	0	['add[0][0]']
max_pooling1d (MaxPooling1D)	(None, 4000, 16)	0	['activation_1[0][0]']
...			
max_pooling1d_4 (MaxPooling1D)	(None, 250, 128)	0	['activation_12[0][0]']
average_pooling1d (AveragePool ing1D)	(None, 83, 128)	0	['max_pooling1d_4[0][0]']
flatten (Flatten)	(None, 10624)	0	['average_pooling1d[0][0]']
dense (Dense)	(None, 256)	2720000	['flatten[0][0]']
dense_1 (Dense)	(None, 128)	32896	['dense[0][0]']
output (Dense)	(None, 5)	645	['dense_1[0][0]']
=====			
Total params: 3,088,597			
Trainable params: 3,088,597			
Non-trainable params: 0			

Ilustración 151: Construcción del modelo.

Entrenar el modelo

```
# Entrenar modelo

#sobreescribo EPOCHS = 1
#EPOCHS = 3
#EPOCHS = 50

history = model.fit(
    train_ds,
    epochs=EPOCHS,
    validation_data=valid_ds,
    callbacks=[earlystopping_cb, mdlcheckpoint_cb],
)
```

Epoch 1/50
16/16 [=====] - 520s 18s/step - loss: 1.9162 - accuracy: 0.3992 - val_loss: 0.5864 - val_accuracy: 0.7725
Epoch 2/50
16/16 [=====] - 13s 677ms/step - loss: 0.6175 - accuracy: 0.7781 - val_loss: 0.4493 - val_accuracy: 0.8693
Epoch 3/50
16/16 [=====] - 13s 689ms/step - loss: 0.3201 - accuracy: 0.8863 - val_loss: 0.2529 - val_accuracy: 0.9321
Epoch 4/50
16/16 [=====] - 13s 673ms/step - loss: 0.2168 - accuracy: 0.9253 - val_loss: 0.2387 - val_accuracy: 0.9202
Epoch 5/50
16/16 [=====] - 12s 649ms/step - loss: 0.2093 - accuracy: 0.9275 - val_loss: 0.2321 - val_accuracy: 0.9287
Epoch 6/50
16/16 [=====] - 12s 662ms/step - loss: 0.1766 - accuracy: 0.9431 - val_loss: 0.2858 - val_accuracy: 0.9066
Epoch 7/50
16/16 [=====] - 12s 657ms/step - loss: 0.0764 - accuracy: 0.9716 - val_loss: 0.1731 - val_accuracy: 0.9508
...
Epoch 27/50
16/16 [=====] - 12s 657ms/step - loss: 0.0764 - accuracy: 0.9716 - val_loss: 0.1731 - val_accuracy: 0.9508
Epoch 28/50
16/16 [=====] - 13s 672ms/step - loss: 0.0660 - accuracy: 0.9754 - val_loss: 0.2166 - val_accuracy: 0.9474
Epoch 29/50
16/16 [=====] - 12s 649ms/step - loss: 0.0360 - accuracy: 0.9843 - val_loss: 0.1528 - val_accuracy: 0.9643
Epoch 30/50
16/16 [=====] - 12s 664ms/step - loss: 0.0329 - accuracy: 0.9877 - val_loss: 0.2981 - val_accuracy: 0.9457

Ilustración 152: Entrenamiento del modelo.

Imprimir las métricas: Accuracy y Loss

```
print(model.evaluate(valid_ds))
```

4/4 [=====] - 2s 314ms/step - loss: 0.1011 - accuracy: 0.9643 [0.10114678740501404, 0.9643463492393494]
--

Evolución de las métricas: Accuracy y Loss por EPOCHS

```
def plot_history(history):
    acc = history.history["accuracy"]
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    val_accuracy = history.history["val_accuracy"]

    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12,5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, "b", label="training_acc")
    plt.plot(x, val_accuracy, "r", label="validation_acc")
    plt.title("Accuracy")

    plt.subplot(1, 2, 2)
    plt.plot(x, loss, "b", label="training_loss")
    plt.plot(x, val_loss, "r", label="validation_loss")
    plt.title("Loss")

plot_history(history)
```

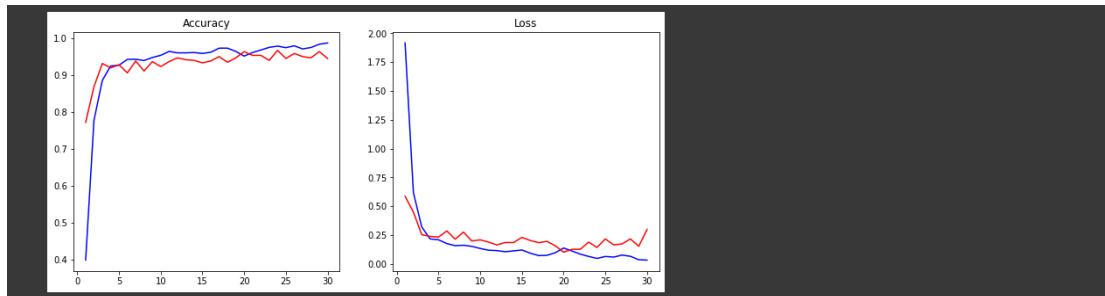


Ilustración 153: métricas: Accuracy y Loss por EPOCHS.

Matriz de Confusión

```
# Matriz de Confusión
# Create x and y tensors

x_valid = None
y_valid = None

for x, y in iter(valid_ds):
    if x_valid is None:
        x_valid = x.numpy()
        y_valid = y.numpy()
    else:
        x_valid = np.concatenate((x_valid, x.numpy()), axis=0)
        y_valid = np.concatenate((y_valid, y.numpy()), axis=0)

# Generate predictions
y_pred = model.predict(x_valid)

# Calculate confusion matrix
confusion_mtx = tf.math.confusion_matrix(
    y_valid, np.argmax(y_pred, axis=-1)
)

# Dibujar la matriz de confusión
plt.figure(figsize=(14, 5))
sns.heatmap(
    confusion_mtx, xticklabels=class_names_process, yticklabels=class_names_process,
    annot=True, fmt="g"
)
```

```

plt.xlabel("Prediction")
plt.ylabel("Label")
plt.title("Validation Confusion Matrix")
plt.show()

for i, label in enumerate(class_names_process):
    if i < confusion_mtx.shape[0]:
        precision = confusion_mtx[i, i] / np.sum(confusion_mtx[:, i])
        recall = confusion_mtx[i, i] / np.sum(confusion_mtx[i, :])
        print("{0:15} Precision:{1:.2f}%; Recall:{2:.2f}%".format(label, precision * 100,
recall * 100))

```

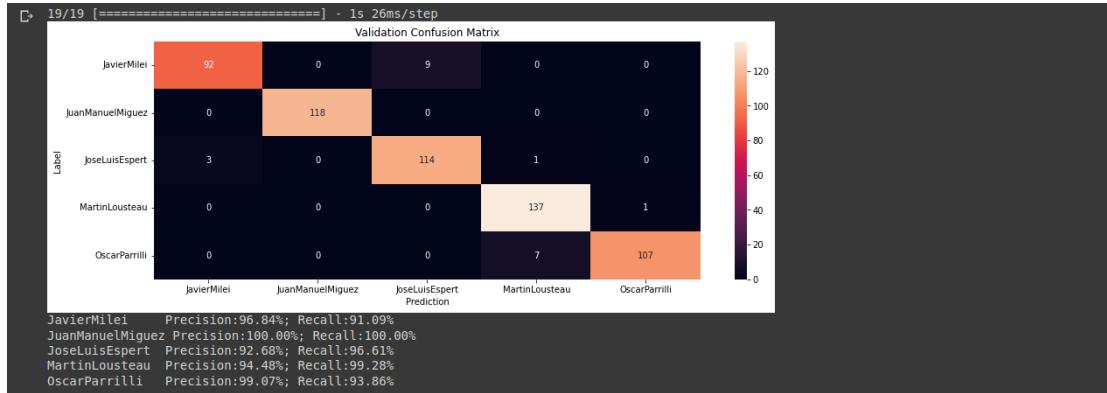


Ilustración 154: Matriz de Confusión.

Verificación del modelo

```

#redefino BATCH_SIZE, por si la cantidad de muestras es menor a 128
BATCH_SIZE = 128
SAMPLES_TO_DISPLAY = 10

print("valid_audio_paths={} \n valid_labels={}".format(valid_audio_paths,
valid_labels))

test_ds = paths_and_labels_to_dataset(valid_audio_paths, valid_labels)

#print("test_ds={}".format(test_ds[-1]))
test_ds = test_ds.shuffle(buffer_size=BATCH_SIZE * 8,
seed=SHUFFLE_SEED).batch(BATCH_SIZE)
test_ds = test_ds.map(lambda x, y: (add_noise(x, noises, scale=SCALE), y))

for audios, labels in test_ds.take(1):
    # Obtener la señal FFT
    fffts = audio_to_fft(audios)
    # Predecir
    y_pred = model.predict(ffffs)
    # Tomar muestras aleatorias
    rnd = np.random.randint(0, BATCH_SIZE, SAMPLES_TO_DISPLAY)

    audios = audios.numpy()[rnd, :, :]
    labels = labels.numpy()[rnd]
    y_pred = np.argmax(y_pred, axis=-1)[rnd]

    print("labels= {}, y_pred={}".format(labels,y_pred))

    cm=confusion_matrix(labels, y_pred)
    print(cm)

    for index in range(SAMPLES_TO_DISPLAY):

```

```

# Para cada muestra, imprimir la etiqueta verdadera y predicha, así como
ejecutar la voz con su ruido
print(
    "Speaker:{} {}{}\tPredicted:{} {}{}\n".format(
        "[92m" if labels[index] == y_pred[index] else "[91m",
        class_names_process[labels[index]],
        "[92m" if labels[index] == y_pred[index] else "[91m",
        class_names_process[y_pred[index]],
    )
)
display(Audio(audios[index, :, :].squeeze(), rate=SAMPLING_RATE))

```

```

In [1]: valid_audio_paths=['/content/drive/MyDrive/proyectoAudio/audio/audio/JavierMilei/163.wav', '/content/drive/MyDrive/proyectoAudio/audio/audio/OscarParrilli/163.wav', '/content/drive/MyDrive/proyectoAudio/audio/audio/JuanManuelMiguez/163.wav', '/content/drive/MyDrive/proyectoAudio/audio/audio/MartinLousteau/163.wav', '/content/drive/MyDrive/proyectoAudio/audio/audio/JoseLuisEsperf/163.wav']
valid_labels=[0, 4, 3, 2, 3, 2, 4, 1, 3, 0, 4, 4, 3, 1, 1, 1, 4, 4, 2, 4, 1, 3, 4, 0, 3, 3, 3, 0, 3, 3, 2, 4, 3, 2, 4, 0, 4, 2, 0, 4/4 [=====] - 0s 25ms/step
labels= [1 3 0 1 4 2 4 4 2 3], y_pred=[1 3 0 1 4 2 4 4 2 3]
[[1 0 0 0 0]
[0 2 0 0 0]
[0 0 2 0 0]
[0 0 0 2 0]
[0 0 0 0 3]]
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: MartinLousteau Predicted: MartinLousteau
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: JavierMilei Predicted: JavierMilei
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: JuanManuelMiguez Predicted: JuanManuelMiguez
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: OscarParrilli Predicted: OscarParrilli
▶ 0:00/0:01 ⏪ ⏴ ⏵ :
Speaker: JoseLuisEsperf Predicted: JoseLuisEsperf
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: OscarParrilli Predicted: OscarParrilli
▶ 0:00/0:01 ⏪ ⏴ ⏵ :
Speaker: OscarParrilli Predicted: OscarParrilli
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: JoseLuisEsperf Predicted: JoseLuisEsperf
...
Speaker: JoseLuisEsperf Predicted: JoseLuisEsperf
▶ 0:01/0:01 ⏪ ⏴ ⏵ :
Speaker: MartinLousteau Predicted: MartinLousteau
▶ 0:00/0:01 ⏪ ⏴ ⏵ :

```

Ilustración 155: Verificación del modelo.

Clasificar con qué categoría es compatible VozViva.wav

```

# Clasificar a qué categoría corresponde el audio VozViva.wav

BATCH_SIZE = 1
#SAMPLES_TO_DISPLAY = 1

valid_audio_paths_vivo = []
valid_audio_paths_vivo += ["{}".format(DATASET_ROOT + '/audio/VozViva/VozViva.wav')]
valid_labels += []
test_ds = paths_and_labels_to_dataset(valid_audio_paths_vivo, [6])
test_ds = test_ds.shuffle(buffer_size=BATCH_SIZE * 8,
seed=SHUFFLE_SEED).batch(BATCH_SIZE)
test_ds = test_ds.map(lambda x, y: (add_noise(x, noises, scale=SCALE), y))
for audios, labels in test_ds.take(1):
    # Obtener la señal FFT
    ffts = audio_to_fft(audios)
    # Predecir
    y_pred = model.predict(ffts)
    max_pred_value=0

```

```

idx_max_pred_value=0
name_max_pred_value=""
pred_name = class_names_process[np.argmax(y_pred)]
print(pred_name)
print("0) -- {} {}".format(pred_name, y_pred) )
for index, value in enumerate(y_pred[0]):
    if (value > 0.1):
        print("y_pred({})<={}>".format(index,value, class_names_process[index]) )
    if(value>max_pred_value):
        max_pred_value=value
        idx_max_pred_value=index
        name_max_pred_value=pred_name
print("1) proba.cercanía=[{}] orden_categoria={}
nombre_categoria={}".format(max_pred_value, idx_max_pred_value,
name_max_pred_value) )
y_pred = np.argmax(y_pred, axis=-1)
print("2) predicción(Voz en Vivo) = {} ... y_pred={}
index={}".format(class_names_process[y_pred[0]], y_pred, index))
print("3) registro (Voz en Vivo) = {} - {}".format(valid_audio_paths_vivo[0],
y_pred, ) )

display(Audio( valid_audio_paths_vivo[0], rate=SAMPLING_RATE ) )

```

The screenshot shows a Jupyter Notebook cell with the following output:

```

In [1]: [=====] - 0s 16ms/step
JuanManuelMiguez
0) -- JuanManuelMiguez [[2.9335742e-11 1.0000000e+00 4.0673992e-11 6.5477962e-13 4.1047493e-13]]
y_pred(1)<=1.0 JuanManuelMiguez
1) proba.cercanía=[1.0] orden_categoria=1 nombre_categoria=JuanManuelMiguez
2) predicción(Voz en Vivo) = JuanManuelMiguez ... y_pred=[1] index=4
3) registro (voz en Vivo) = /content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav - [1]

```

Below the code cell is a waveform visualization with a play button and a progress bar.

Ilustración 156: Clasificación.

Dibujar la forma de onda y sonograma de VozViva.wav

```

# Dibujar forma de onda y espectrograma del audio: VozViva.wav

# Read the wav file (mono)
samplingFrequency, signalData =
wavfile.read('/content/drive/MyDrive/proyectoAudio/audio/VozViva/VozViva.wav')

# Plot the signal read from wav file
plt.rcParams["figure.figsize"] = [10.00, 8.00]
plt.rcParams["figure.autolayout"] = True
plt.subplot(211)
plt.title('Wave')
plt.plot(signalData)
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.subplot(212)
plt.title('Spectrogram')
plt.specgram(signalData,Fs=samplingFrequency)
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.show()

```

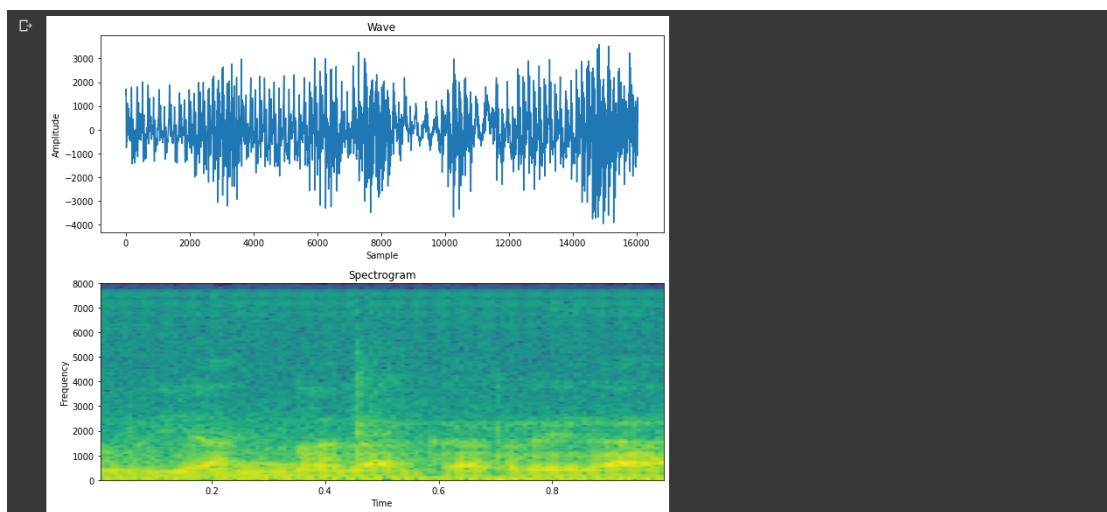


Ilustración 38: oscilograma y sonograma de una prueba.