UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

USO DE *MACHINE LEARNING* Y PROCESAMIENTO DE SEÑALES PARA DETECCIÓN DE COMERCIALES EN LA RADIO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

DIEGO IRARRÁZAVAL INFANTE

PROFESORA GUÍA:
ANTONIA LARRAÑAGA RECART.

MIEMBROS DE LA COMISIÓN:
FRANCISCO RIVERA SERRANO
JUAN MANUEL BARRIOS NÚÑEZ

Este trabajo ha sido parcialmente financiado por:
UNHOLSTER

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL
ELÉCTRICO
POR: DIEGO IRARRÁZAVAL INFANTE
FECHA: 2022
PROF. GUÍA: ANTONIA LARRAÑAGA RECART

USO DE *MACHINE LEARNING* Y PROCESAMIENTO DE SEÑALES PARA DETECCIÓN DE COMERCIALES EN LA RADIO

El problema de clasificación de audio presenta distintos entornos de aplicación. Desde predicción de instrumento musical, identificación de especie de animal en base al sonido emitido, y otros. Aún con la variedad de problemas de clasificación de audio, la bibliografía respecto la detección de comerciales es escasa. Muchos enfoques se proponen de todas formas para el problema general. Los más modernos implican la extracción de espectrogramas de las muestras y clasificación con modelos de redes neuronales.

Esta memoria se centra en la detección de comerciales en un subconjunto de radio emisoras de Chile. Se entenderá por comercial aquellas grabaciones de publicidad creadas por las distintas compañías, no aquellas menciones de los locutores a los auspiciadores. Las radios corresponden a: Bíobio, Pudahuel, Cooperativa, Corazón, El Pinguino, Fiesta, Imagina, Luna Tropical y Valparaíso. La elección es debido a la disponibilidad de grabaciones de las radios recién mencionadas.

En el presente trabajo se crea un *dataset* compuesto por más de 1200 minutos de audio correspondientes a grabaciones de correspondientes a las emisoras ya mencionadas. Este *dataset* generado se utilizó para entrenar distintos modelos de clasificación, agrupados en tres enfoques distintos.

Los modelos y enfoques implementados son los siguientes: *a*) espectrogramas (generar imágenes a partir de segmentos de audio) para generación de características y redes neuronales convolucionales como modelo de clasificación, *b*) extraer coeficientes de MEL utilizando análisis de ventana de distintos tamaño para implementar modelos más simples con `sk-learn` para clasificar y, *c*) redes neuronales recurrentes (mediante la librería de procesamiento de audio *Resemblyzer*) para la extracción de características y los modelos simples de `sk-learn` como clasificador.

El modelo que se elige para ser utilizado en ambientes de producción corresponde a utilizar *Resemblyzer* para extracción de características y modelos de clasificación de `sk-learn`. Los resultados obtenidos en producción con audios de la radio Bíobio son de un 90 % de *accuracy* sobre un conjunto de 3 hrs y 20 minutos de duración.

Finalmente, se detalla sobre la estructura e infraestructura AWS utilizada para el paso a producción de los modelos desarrollados.

Para Mary, Damián

y Ana.

Agradecimientos

En primer lugar, quiero agradecer a Antonia por su paciencia, comprensión y consejo durante el desarrollo. Durante todo el proceso de desarrollo y escritura de la tesis estuvo a total disposición.

Junto a Antonia, agradezco a todo Unholster por poner a disposición datos, recursos computacionales y por permitirme realizar la tesis mientras trabajaba.

Luego, quiero agradecer a mi mamá, Emiliana. Sin ella, esta empresa hubiese sido imposible. Su perseverancia e insistencia, muchas veces no bien recibida de mi parte, fueron fundamental impulso para completar este trabajo. De igual forma, toda mi familia y amigos que apoyaron en el proceso.

Por último, gracias a los miembros de la comisión que hacen posible este proceso de titulación.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

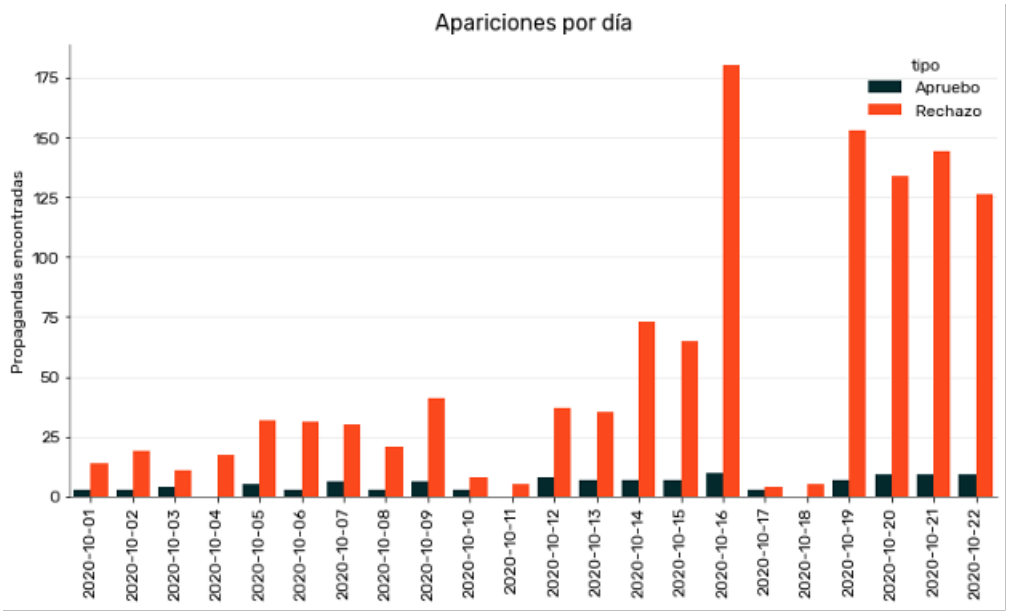
Introducción

1.1. Antecedentes Generales

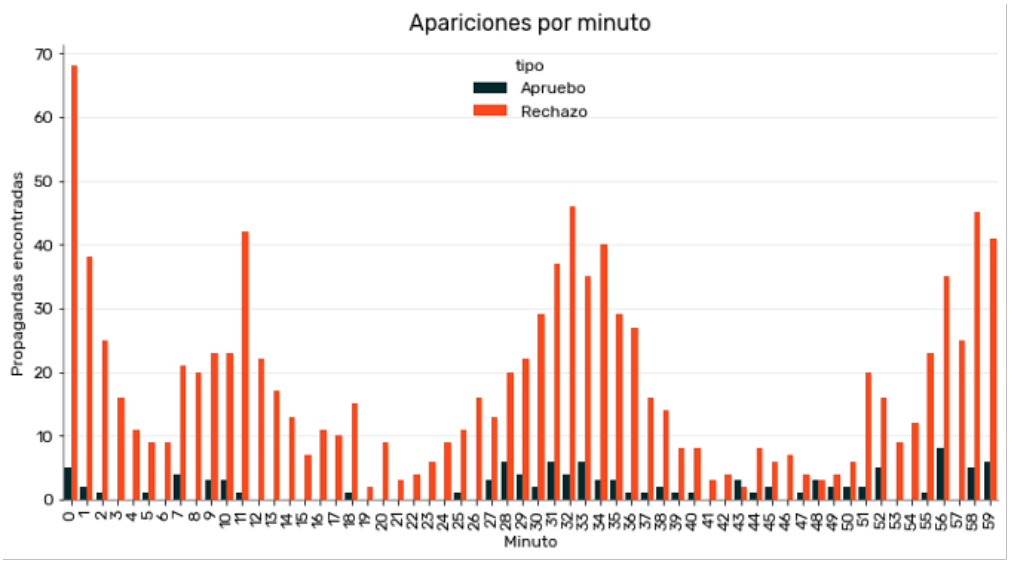
Unholster es una empresa de tecnología experta en desarrollo de software a medida, datos, analítica predictiva y en ayudar a clientes con sus problemas y proyectos tecnológicos. En distintas ocasiones, Unholster ha prestado distintos tipos de servicios de consultoría a equipos de campañas política y se estudian con particular interés todos los procesos de elecciones, mediante la plataforma propia decidechile.cl.

En esa línea, para el plebiscito de Noviembre del año 2020, la empresa (Unholster, 2020) realizó un estudio para conocer cuándo (en que momento del día y la semana) y dónde (qué radio emisoras) estaban las publicidades en las radios de las campañas políticas (apruebo y rechazo). Para esto se implementó un programa en base a huella digital acústica (en adelante, *fingerprints*) que a partir de una grabación de publicidad grabada y procesada (se utilizará el término audio maestro, o maestro para esto), buscaba las ocurrencias de esta en grabaciones de la radio.

En la figura 1.1 obtenida de Unholster (2020), se observan los resultados del estudio realizado por Unholster que motiva este trabajo. En estos se observa la distribución de publicidad de apruebo o rechazo, agrupados según la fecha que ocurrieron, la hora del día y el minuto que corresponden dentro de una hora de transmisión:



(a) Distribución de comerciales por día.



(b) Distribución de comerciales por minuto.

Figura 1.1: Patrones de apruebo y rechazo en la propaganda del plebiscito en las radios.

Para la búsqueda de los datos que permitieron escribir el artículo, se debió encontrar manualmente publicidades correspondientes al apruebo o al rechazo. Esto significó escuchar constantemente la radio y estar atento a la publicidad que se reproducía. Cuando lo transmitido correspondía a una publicidad del plebiscito, se tomaba nota de la hora y la radio emisora para luego buscar ese instante en las grabaciones y recortar el segmento indicado de comercial. Cada segmento correspondiente a los distintos comerciales corresponde a un maestro o audio maestro.

De esta forma surgen las primeras intenciones de desarrollar un modelo de inteligencia artificial que sea capaz de encontrar publicidades en la transmisión de la radio, sin necesariamente identificarlo como político o no. Este desarrollo permite limitar el horizonte de

búsqueda de publicidades políticas y, en segunda instancia, la implementación de distintos tipos de reportes e informes respecto de la distribución de publicidad (no sólo política) en la radio.

1.2. Situación actual de búsqueda de maestros

La situación actual, según lo descrito en los párrafos anteriores, corresponde a un proceso manual en el que se involucró en primera instancia a toda persona de la oficina que quisiere participar en un concurso de encontrar publicidades políticas. Esta metodología tiene una gran ventaja: al ser los comerciales identificados por personas, hay (en teoría) un 100 % de precisión. Por otro lado, la cobertura en variedad de horarios y radios es baja, además de ser un proceso lento y tedioso.

Adicionalmente a los estudios de distribuciones de publicidades del plebiscito (o cualquier votación futura), se quiere estudiar el uso de la radio como plataforma publicitaria. Para esto, se espera que con los resultados obtenidos por el modelo, se puedan construir visualizaciones con la distribución de publicidades en la radio. Esto con las metodologías actuales es irrealizable, ya que no permite encontrar o identificar publicidades nuevas, porque la búsqueda se basa en similitud con los maestros ya encontrados. Es decir, si no se tiene una muestra del audio a buscar no se puede realizar la búsqueda.

Dado lo anterior, y motivados por la existencia de distintos algoritmos y *datasets* enfocados en la clasificación de audio, se propone como tema implementar un clasificador de comerciales en la radio.

1.3. Objetivos

1.3.1. Objetivos Generales

Implementar un programa en *Python* que utilice herramientas de *machine learning* y procesamiento de señales para detectar comerciales en la radio. Este debe ser capaz de detectar comerciales que no hayan sido escuchados anteriormente. Es decir, que una vez entrenado y desarrollado el modelo puede clasificar y detectar comerciales fuera de la base de entrenamiento y prueba. Además, el modelo debe ser al menos tan rápido en tiempo de ejecución como la solución actual para la detección de comerciales para funcionar en ambientes de producción. Sobre esto, se desarrollan los objetivos específicos.

1.3.2. Objetivos Específicos

Los objetivos específicos se enumeran a continuación:

1. Formular el problema como uno que se pueda resolver utilizando herramientas de aprendizaje de máquinas.

2. Creación de un dataset que permita entrenar diferentes modelos de inteligencia artificial. Este debe estar compuesto de segmentos de grabaciones de radio y una etiqueta única.

3. Entrenamiento y evaluación de modelos de inteligencia artificial, para lo cual se proponen tres enfoques distintos en base a la combinación del clasificador con el método para extraer características:

   a) Espectrogramas de MEL y CNN

   b) Coeficientes de MEL y modelos de clasificación

   c) Resemblyzer y modelos de clasificación

4. Proponer métrica para evaluar la eficiencia en tiempo de ejecución del modelo desarrollado. Utilizarla para la evaluación de los modelos.

5. Puesta en marcha: el modelo desarrollado será utilizado en entorno de producción para la obtención de comerciales. Es decir, se debe implementar un código simple de utilizar que entregue los resultados de forma ordenada para la búsqueda de maestros y la generación de reportes de distribución de comerciales en la radio.

1.4. Estructura del trabajo

El trabajo está estructurado en 6 capítulos:

- **Capítulo 1 Introducción:** antecedentes generales, motivación y objetivos.
- **Capítulo 2 Marco Teórico y Estado del Arte:** fundamentos técnicos necesarios para comprender los distintos tópicos tratados. Desde representaciones de audio hasta modelos de clasificación con aprendizaje de máquinas. Revisión de modelos y herramientas existentes relacionados a esta memoria.
- **Capítulo 3 Metodología y Datos:** se presentan las herramientas utilizadas para el desarrollo del modelo y el cumplimiento de los distintos objetivos específicos. También se detalla sobre la creación del dataset que permite el entrenamiento de modelos y las métricas a utilizar para la evaluación del modelo.
- **Capítulo 4 Implementación:** se detallan los distintos enfoques y modelos desarrollados a lo largo del proceso.
- **Capítulo 5 Resultados:** resultados respecto de los modelos desarrollados en el capítulo 4 y los correspondientes análisis.
- **Capítulo 6 Conclusión:** conclusiones y trabajo futuro.

Capítulo 2

Marco Teórico y Estado del Arte

2.1. Marco Teórico

2.1.1. Representaciones del audio

Para la implementación de un modelo de clasificación de audio, en primer lugar, se deben generar representaciones de este. Esto permite utilizar algoritmos de ML conocidos e implementar una gran variedad de modelos y soluciones al problema planteado.

En la presente sección se detallan distintas representaciones. La primera parte corresponde a métodos clásicos (transformar al dominio de las frecuencias y obtener espectrogramas) para luego, introducir una representación de audio basada en Redes Neuronales.

2.1.1.1. Forma de onda

La representación en forma de onda corresponde a la amplitud de la señal en función del tiempo. La figura 2.1 muestra la visualización de esta representación de audio con un ejemplo de fragmento de comercial ([1]). La imagen corresponde a un segmento de audio de 5 segundos muestreado a $16[kHz]$.

[1] Elaborado con *python* a partir de un audio del *dataset* creado.

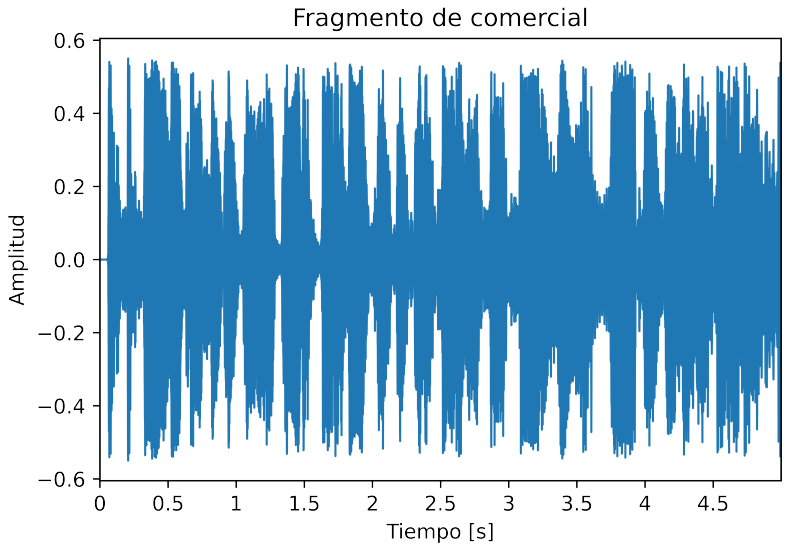


Figura 2.1: Señal de audio: amplitud en el tiempo. Elaboración propia.

Esta representación contiene información sólo de la amplitud de la señal en función del tiempo. Es decir, la información de la intensidad del sonido en el tiempo. Quiere decir que los ruidos, cercanía o lejanía a un micrófono e interferencias de la radio afectan gravemente esta representación. A partir de esta representación, se construirán las representaciones en el espacio de las frecuencias.

2.1.1.2. Representaciones en frecuencias

La representación en el espacio de las frecuencias corresponde al resultado de aplicar la transformada de Fourier a la señal. La transformación de una señal discreta (como la representación de la figura 2.1) a su representación en las frecuencias se realiza con la transformada de Fourier discreta. Esta se define por la siguiente expresión matemática:

Dada una señal discreta $x[n]$,

$$X_{2\pi}(\omega) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-i\omega n} \tag{2.1}$$

La figura 2.2 ayuda a comprender como una señal se representa en el espacio de las frecuencias:

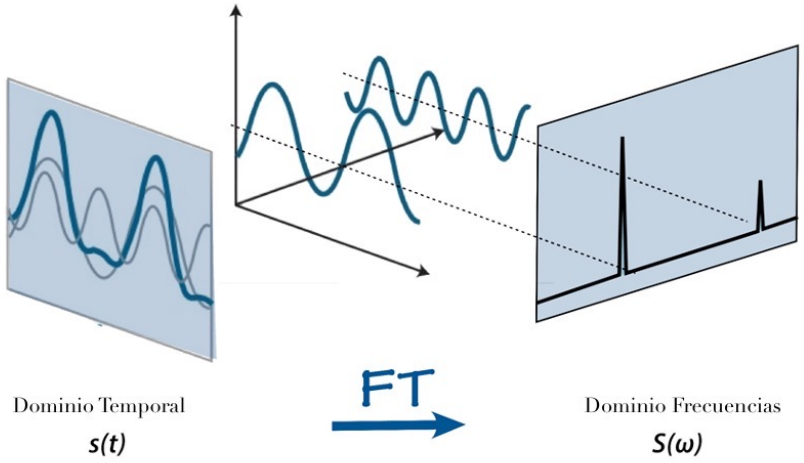


Figura 2.2: Esquema de la transformada de Fourier. Figura obtenida de https://aavos.eu/glossary/fourier-transform/.

El objetivo de esta representación en frecuencias de la señal es obtener información que muestra información adicional a la contenida en la amplitud de la señal. En la figura 2.3 se observa la magnitud de los coeficientes obtenidos al aplicar esta operación al fragmento de comercial mencionado anteriormente:

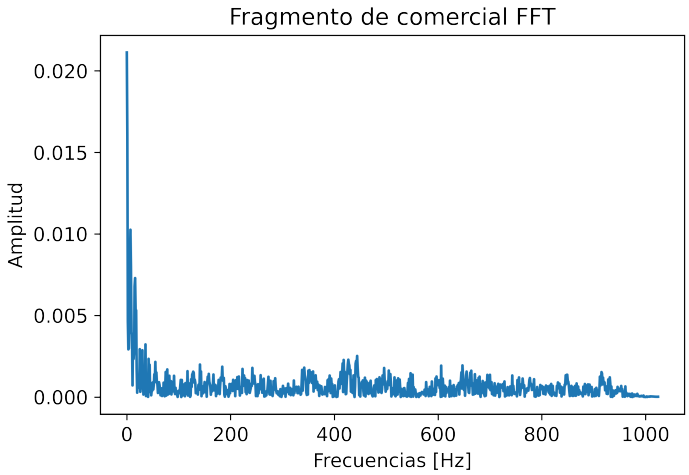


Figura 2.3: Transformada de Fourier del fragmento de comercial. Figura obtenida de https://www.mathworks.com/help/dsp/ref/dsp.stft.html y traducida al español.

Esta representación es útil en ciertas aplicaciones pero tiene una desventaja importante: se pierde la información temporal contenida en la señal. Independiente del largo del audio, se obtiene sólo una representación para este. Para obtener información acerca de las componentes en frecuencias a través del tiempo, se introducen los espectrogramas.

2.1.1.3. Espectrograma

El espectrograma corresponde a una representación de la variación de la intensidad de las componentes en frecuencias de una señal en el tiempo. Esto es, cómo varía la intensidad de cada frecuencia en el tiempo.

Para obtener la información de las frecuencias en el tiempo, se realiza un análisis con ventanas deslizantes que calculan para un segmento (ventana) de la señal la transformada de Fourier y se concatenan los resultados. La figura 2.4 es una representación de cómo se obtiene un espectrograma utilizando ventanas deslizantes:

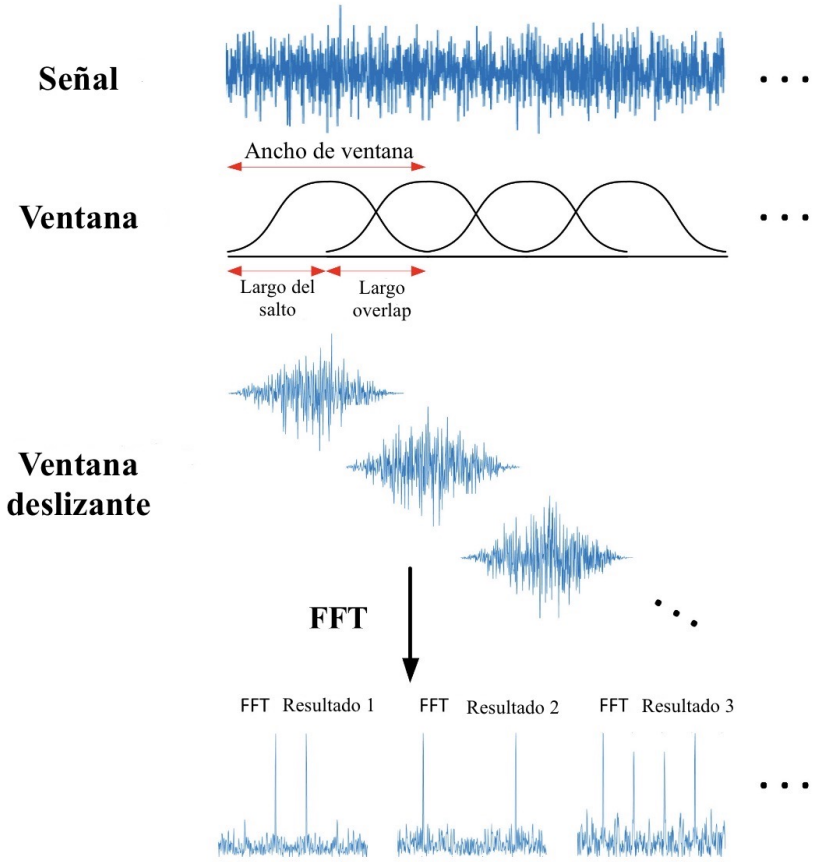


Figura 2.4: Espectrograma del fragmento de comercial.

La figura 2.5 muestra la visualización de un espectrograma aplicado al mismo segmento de comercial que se observa en la figura 2.1.

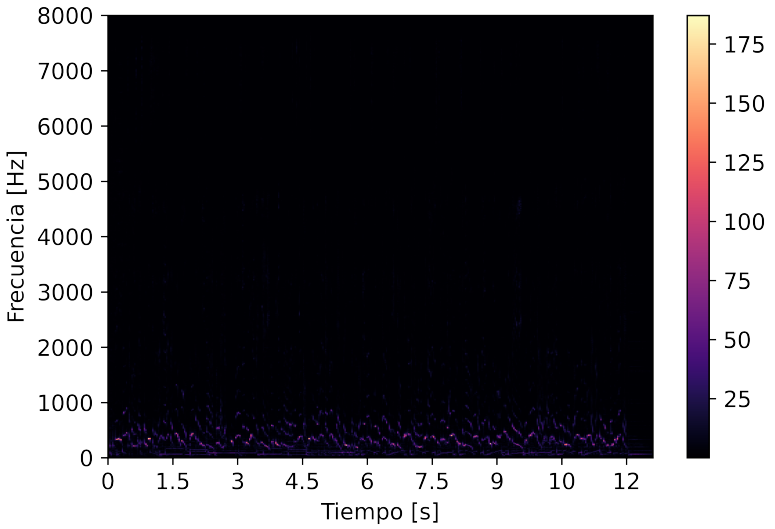


Figura 2.5: Espectrograma del fragmento de comercial.

Es difícil observar en la imagen anterior alguna frecuencia que destaque. Esto se debe a que las frecuencias normalmente producidas por humanos y, las que mejor somos capaces de escuchar, se encuentran en el rango de $[0, 2000]Hz$. Para mejorar la percepción en este rango se introducen las siguientes representaciones.

2.1.1.4. Escalas logarítmicas y coeficientes de MEL

Para obtener de mejor manera tanto la información contenida en el rango $[0, 2000]Hz$ como en el resto del espectro, se aplican distintas transformaciones al espectrograma: transformación logarítmica a la intensidad del sonido y una escala en el eje de las frecuencias.

La primera, corresponde a escalar la intensidad del sonido a decibeles. La visualización de esta transformación se observa en la figura 2.6

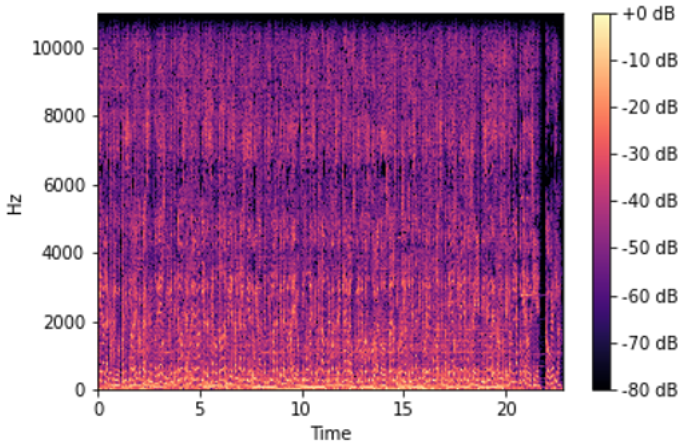
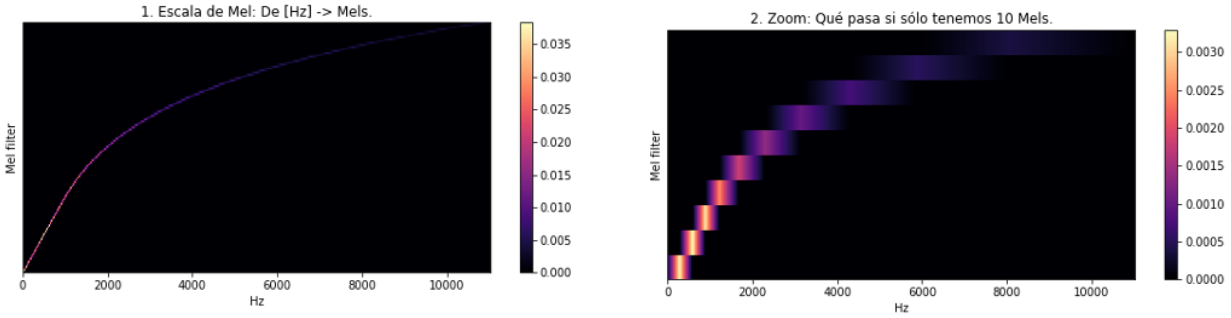


Figura 2.6: Espectrograma con intensidad en escala de decibeles.

Respecto a la figura 2.6 se debe mencionar que la escala usada es en relación a la máxima potencia de la señal. Esto quiere decir que $0[dB]$ representan el máximo de potencia de la señal procesada.

Para este trabajo, basado también en Collobert, Puhrsch, y Synnaeve (2016), Guzhov, Raue, Hees, y Dengel (2020), Hershey et al. (2017) y Palanisamy, Singhania, y Yao (2020), se propone el uso de la escala de MEL. Esta escala se propone a finales de los 30 (entre 1937 y 1940) y su nombre proviene de *melody*. Su propósito es traducir los valores obtenidos en el espacio de las frecuencias a una escala más comprensible para las personas. En la figura 2.7, se muestra el funcionamiento de la escala. Para generar los gráficos se utilizó *Python*:



(a) Escala de Mel con 128 coeficientes.

(b) Escala de Mel con 10 coeficientes.

Figura 2.7: Escala de Mel. En el eje x se tiene la frecuencia en $[Hz]$ y en el eje y se encuentran los coeficientes.

De esta forma, se obtiene una representación que da mayor importancia a las frecuencias bajas (entre $20[Hz]$ a $2000[Hz]$), que son aquellas que el oído humano mejor entiende y que las personas producen.

En la figura 2.8 se observa la visualización de un espectrograma con coeficientes de MEL:

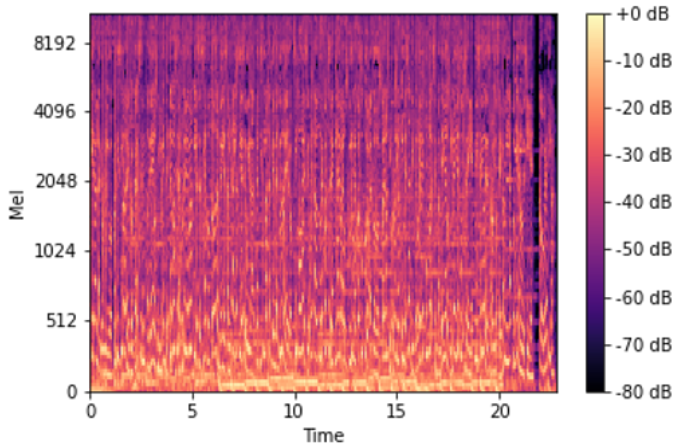


Figura 2.8: Espectrograma de MEL de fragmento de comercial.

2.1.2. Aprendizaje de máquinas

Aprendizaje de Máquinas o *Machine Learning* (en adelante, ML) corresponde a un área de estudio de la Inteligencia Artificial. El objetivo es desarrollar algoritmos que sean capaces de aprender. Aprender significa que mejore su desempeño en alguna tarea específica a partir de la experiencia obtenida en el proceso de entrenamiento. Todos estos conceptos se verán con mayor profundidad en esta sección.

Al final del proceso de entrenamiento, se espera que estas técnicas o algoritmos sean capaces de resolver problemas o tomar decisiones sin que hayan sido programados explícitamente para esta. La naturaleza de estas tareas es variada, desde clasificación de imágenes entre perros y gatos hasta modelos complejos que entienden y escriben texto en base a preguntas de usuarios.

Dependiendo de como se entregan los datos y como los algoritmos aprenden, estos se pueden dividir en cuatro categorías principales: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semisupervisado y aprendizaje reforzado.

En particular para este trabajo se utilizan y entrenan modelos de aprendizaje supervisado. Así mismo, gran parte de la bibliografía revisada se centra en este enfoque, dado el planteamiento del problema.

2.1.2.1. Aprendizaje Supervisado y Clasificación

Corresponde a la tarea de encontrar una función a partir un conjunto de pares *{dato, etiqueta}*, (X, Y) que sea capaz de, dado un nuevo dato X_{input} desconocido, generar una predicción $y_{predicción}$ lo más cercana posible al valor real Y_{real}. Para esto, se utiliza un proceso de entrenamiento en el cual el algoritmo recibe estos pares *{X, Y}* ajustando los parámetros de este en función de una métrica que compara lo obtenido por el modelo con la realidad

(error).

En la figura 2.9, se muestra una representación del proceso de entrenamiento a partir del cual un algoritmo de aprendizaje supervisado mejora el desempeño:

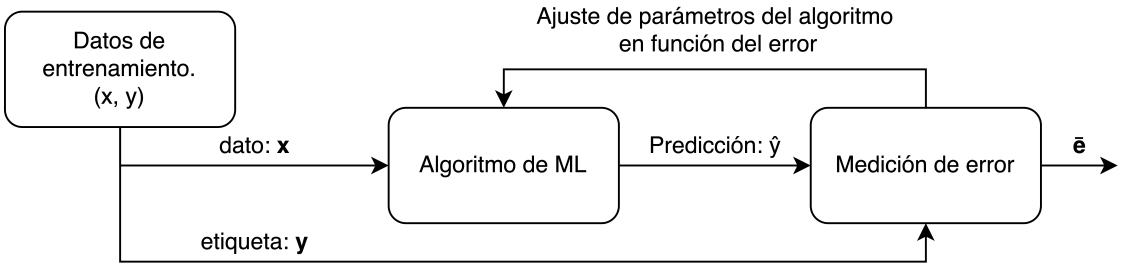


Figura 2.9: Entrenamiento de algoritmos de aprendizaje supervisado. Elaboración propia.

Tomando en cuenta lo anterior, el problema que se plantea en esta memoria se clasifica como uno de aprendizaje supervisado; en particular un escenario de clasificación. En adelante, se revisarán algunos de los algoritmos que se implementan en las distintas soluciones.

2.1.2.2. Algoritmos de *Machine Learning*

En esta sección se revisarán algunos de los algoritmos más usados para clasificación en problemas de *data science* debido a su fácil implementación con librerías como `sk-learn`, presentada por Pedregosa et al. (2011):

2.1.2.2.1. Support Vector Machine

Este algoritmo (en adelante SVM) de clasificación o regresión, se utiliza comúnmente en conjuntos de datos no muy grandes que tienen un determinado espacio de características. Es decir, usualmente cada par $\{dato, etiqueta\}$ está compuesto por un vector de dimensión k fija y una etiqueta.

El proceso de entrenamiento de este algoritmo consiste en encontrar un hiperplano que separe en el espacio de las características, de la mejor forma posible, los puntos (el conjunto de datos) que corresponden a *etiquetas* distintas. Los puntos del conjunto de entrenamiento que se utilizan para dibujar el hiperplano se conocen como vectores de soporte. La figura 2.10 ilustra el algoritmo recién descrito intentando separar los puntos o datos de la clase + de aquellos perteneciente a la clase -.

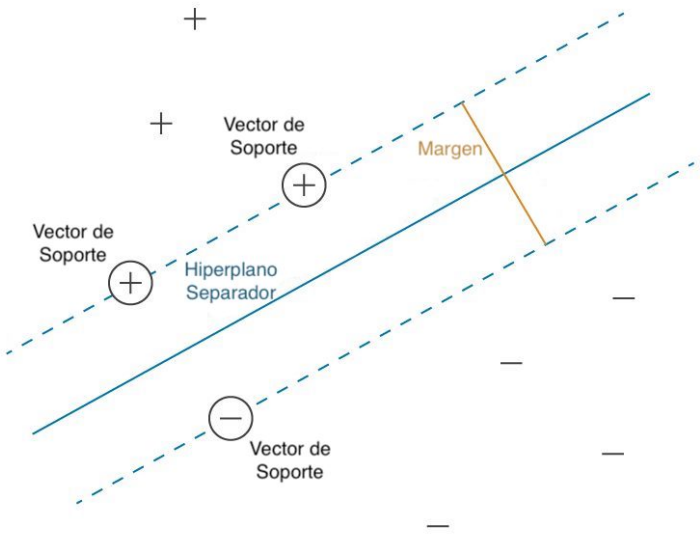


Figura 2.10: Descripción del algoritmo *Support Vector Machine*. Imagen obtenida de https://la.mathworks.com/discovery/support-vector-machine.html.

Uno de los hiperparámetros más importantes que se puede configurar en SVM corresponde al *kernel*. Este, coloquialmente, indica cuál es la forma que puede tener el hiperplano que separa una clase con la otra. Es decir, es lo que determina en que espacio se representarán los datos y puntos para luego ser separados por un hiperplano. Así, problemas que no son fácilmente separables con un *kernel* lineal, si pueden ser separables con uno sigmoide.

2.1.2.2.2. Árboles de decisión

Los árboles toman decisiones a partir de datos históricos (conjunto de entrenamiento), construyendo condiciones para llegar desde la raíz hasta una hoja. En la figura 2.11 (similar al presentado por Mollas, Tsoumakas, y Bassiliades (2019)) se observa un ejemplo simple donde a partir de cuatro características (f_1, f_2, f_3 y f_4), se obtienen los umbrales que toman decisiones y al llegar a las hojas, se entrega una predicción.

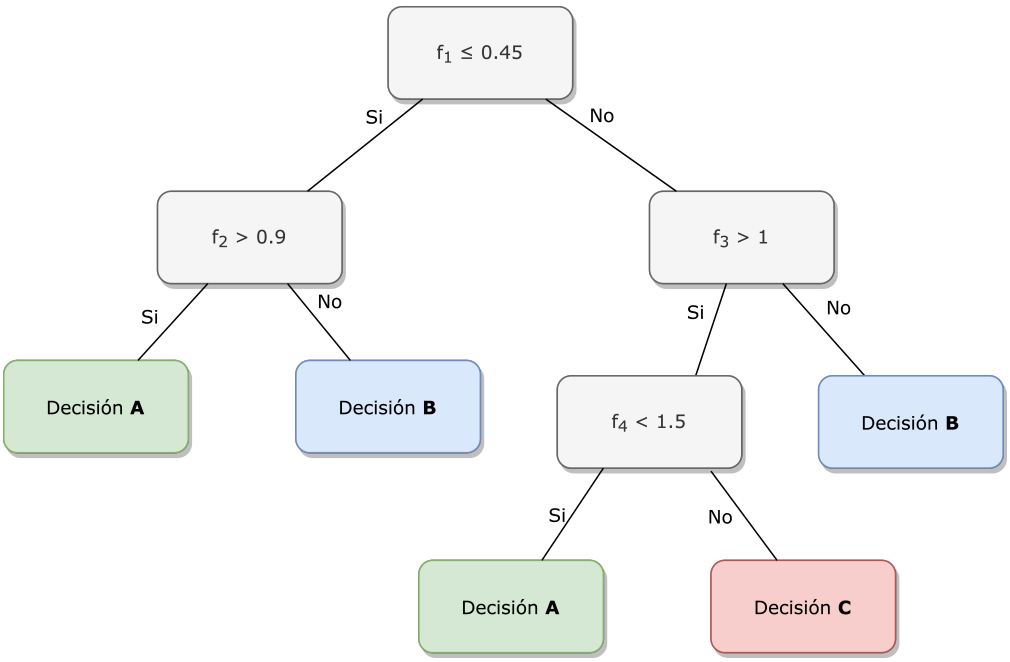


Figura 2.11: Esquema de árbol de decisión.

El principal hiperparámetro de este modelo corresponde a la profundidad del árbol. Mientras más profundo el modelo, se vuelve mejor en el conjunto de entrenamiento pero es susceptible a *overfitting*, esto es, pierde la capacidad de generalizar.

Estos modelos son utilizados en problemas en donde se requiere capacidad de interpretación de los resultados. Existen librerías que permiten visualizar los distintos umbrales y distintas condiciones de decisión de cada hoja, resultando en un algoritmo que es interpretable por las personas.

2.1.2.2.3. *Random Forest*

Random Forest (en adelante RF) corresponde a un algoritmo de ensamble, es decir, RF corresponde a una combinación de árboles de decisión entrenados con los mismos datos pero de forma independiente de los que luego sus predicciones son promediadas. La figura 2.12 ilustra como un conjunto de árboles genera predicciones que luego, bajo cierto criterio, se agrupan en una final.

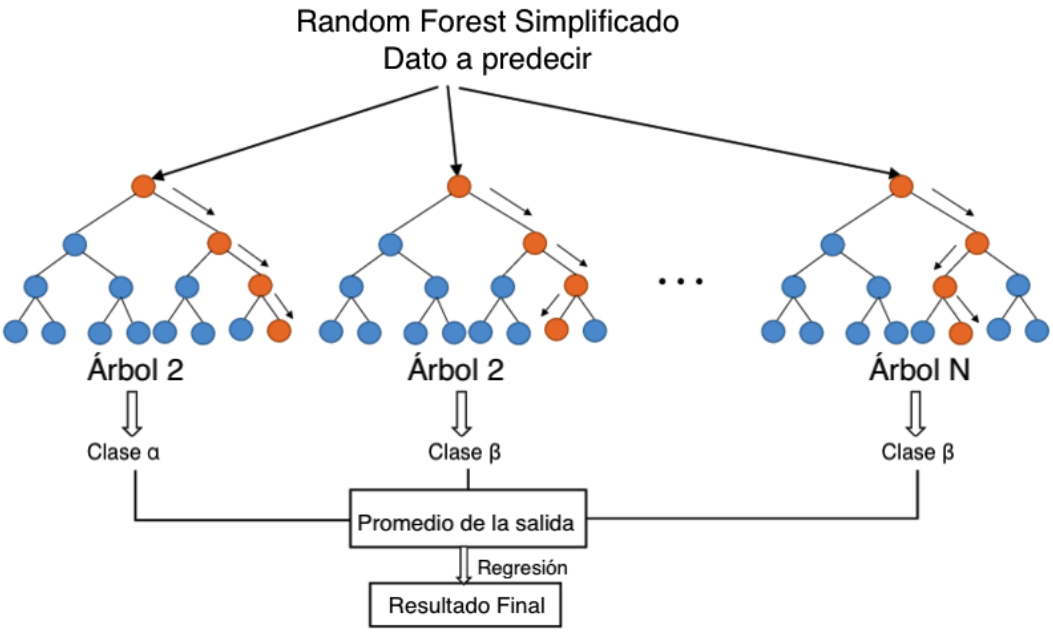


Figura 2.12: Ilustración del funcionamiento de *Random Forest*. Figura obtenida de https://www.researchgate.net/figure/Random-Forest-Simplified-2-Extreme-Random-Forest-Then-Limit-Tree-regression-model-is_fig2_348965642

2.1.2.3. Redes Neuronales

Una red neuronal es un modelo que intenta emular la forma en la que el cerebro humano procesa información. Consiste en un conjunto de neuronas o perceptrones conectados entre sí. La unidad básica, el perceptrón, realiza distintas operaciones para combinar y transformar la información que está en la entrada de esta y entrega un sólo resultado final. La figura 2.13 muestra una representación de un perceptrón:

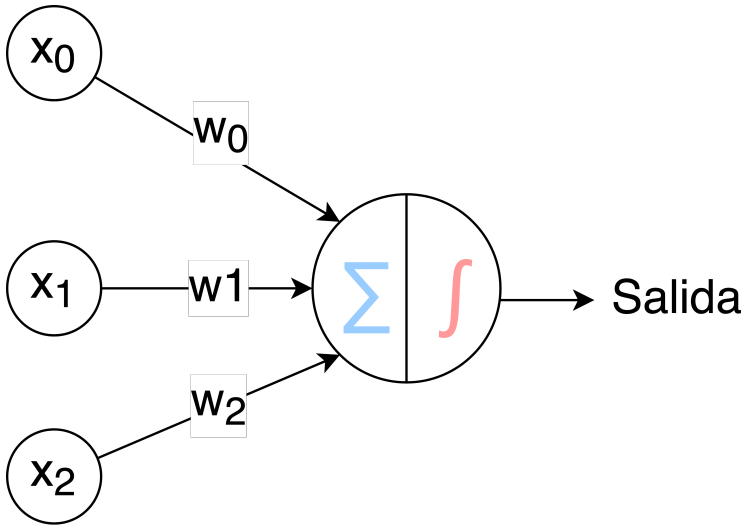


Figura 2.13: Perceptrón: x_i representa los valores de entrada, w_i los pesos sinápticos y $\int$ la función de activación

El diagrama de la figura 2.13 puede ser representado como una expresión matemática. En esta, $X = \{x_0, x_1, x_2, ..., x_n\}$ corresponde a un dato en la entrada $W = \{w_0, w_1, w_2, ..., w_n\}$ corresponde al vector de pesos sinápticos de esa neurona y $\int$ es la función de activación. Así, se escribe la salida y de un perceptrón como la siguiente función:

$$y = f(\mathbf{w} \cdot \mathbf{x}) \tag{2.2}$$

Al organizar más de un perceptrón en capas y conectar la salida de una capa i con la siguiente $i + 1$, se construye una red neuronal, como se ve en la figura 2.14:

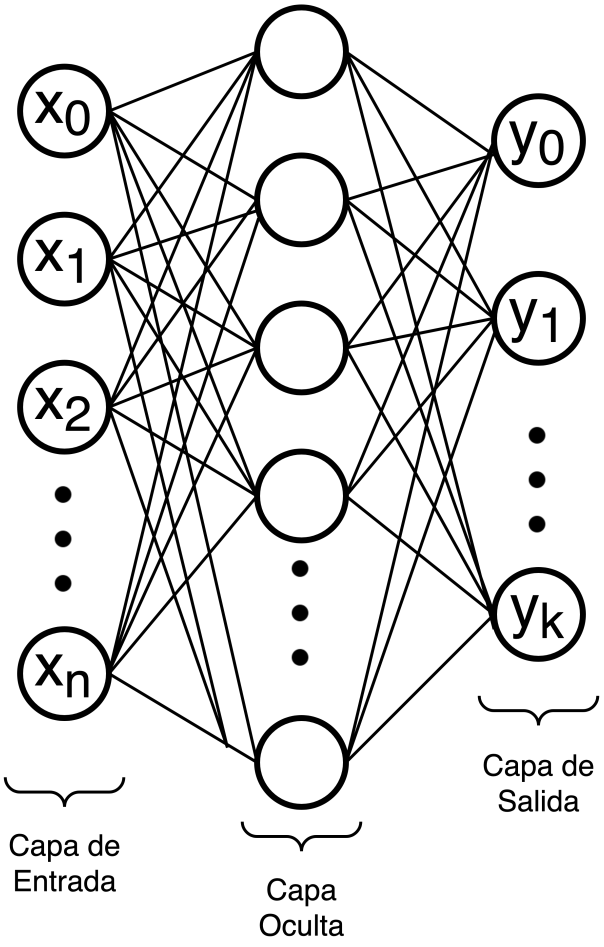


Figura 2.14: Red neuronal tipo *feed forward*.

La expresión de la ecuación 2.2 se puede expandir para representar la red *feed forward* de la siguiente forma. Dado $X = \{x_0, x_1, x_2, ..., x_n\}$ un dato en la capa de entrada, W una matriz de dimensiones (n, j) (j representa la cantidad de neuronas de la capa oculta en la imagen) con los pesos sinápticos y una función de activación f:

$$\vec{y} = f(\mathbf{X}^t \cdot \mathbf{W} + b) \tag{2.3}$$

La ecuación 2.3 introduce un término adicional que corresponde a b o *bias*. Este es un término numérico que se agrega como parte de la operación lineal a la entrada de la función de activación. Es otro parámetro entrenable, al igual que todos los pesos sinápticos W.

Esta función de activación mencionada anteriormente es la que define la salida de cada neurona. Agrega una componente de no-linealidad a la función descrita en 2.3. Su equivalente biológico corresponde al potencial de activación de una neurona. En su versión más básica, es una función que entrega exactamente lo que haya en la entrada ($y = \mathbf{X}^t \cdot \mathbf{W} + b$) si $y \geq 0$, sino el valor de la salida es 0. Las funciones de activación más conocidas corresponden a la función lineal, función escalón, sigmoidal, tangente hiperbólica, función rectificador y *softmax*.

El objetivo o tarea que cumplen las redes neuronales es el aproximar una función f que depende de los datos de entrada x y los parámetros de la red neuronal θ al valor real $\hat{y}$ correspondiente a los datos respectivos, es decir, $y = f(x, \theta)$. Para encontrar esta función, el proceso de entrenamiento itera siguiendo el esquema de la figura 2.9.

En el proceso de entrenamiento, existe una función encargada de medir el desempeño de la red neuronal. Para esto, la función de pérdida compara el resultado obtenido de la red neuronal y con la etiqueta o valor real $\hat{y}$ y entrega esta diferencia al algoritmo de entrenamiento para ajustar los pesos sinápticos de forma inteligente.

Existen distintos tipos de funciones de pérdida dependiendo del problema: si es clasificación binaria o con múltiples clases o si es una regresión. La función de pérdida utilizada en el entrenamiento en este trabajo corresponde a la entropía cruzada:

$$loss = -\sum_{c=1}^{M} y_{o,c} ln(p_{o,c})$$

Donde:

- M corresponde a la cantidad de clases
- $y_{0,c}$ es la indicatriz si la clase c es la correcta para la predicción o (0 si la predicción es incorrecta, 1 si la predicción es correcta)
- $p_{o,c}$ probabilidad de que la predicción o corresponda a la clase c

La unión de una serie de capas de redes neuronales en las que cada neurona de una determinada capa está conectada con todas las neuronas de la capa anterior y la siguiente, forman una red *fully conected*. La figura 2.15 ilustra como se interconectan las capas *fully connected*:

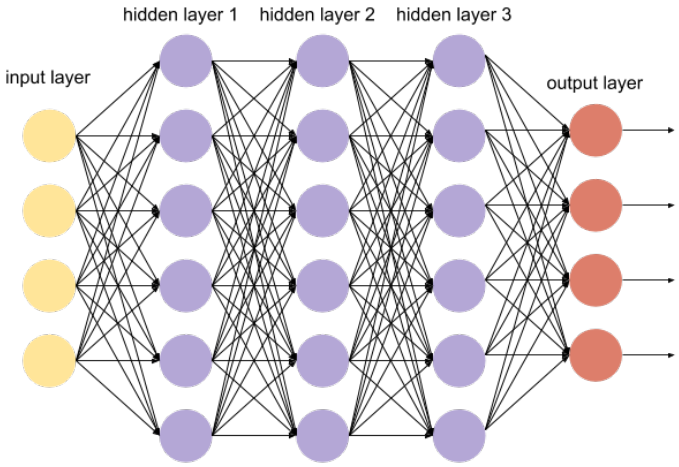


Figura 2.15: Ejemplo una capa *fully conected*. Los puntos morados corresponden a las capas *Fully Conected*

2.1.2.3.1. Deep Learning

Deep Learning (en adelante DL) corresponde a un área de ML basado en la implementación de distintos tipos de redes neuronales. La principal diferencia con los paradigmas convencionales de ML es que las redes neuronales profundas aprenden representaciones intermedias de información por ellas mismas, es decir, el trabajo de extracción de características es aprendido por el mismo algoritmo.

La irrupción de DL en el mundo del aprendizaje de máquina significó un quiebre en el desempeño de los modelos y las competencias en las que se presentaba el estado del arte. La figura 2.16 muestra como la irrupción de DL en el problema conocido como `ImageNet` Deng et al. (2009a), en particular la arquitectura de redes convolucionales AlexNet introducida en Krizhevsky, Sutskever, y Hinton (2012), mejora en 10 puntos el error obtenido en clasificación de imágenes marcando un precedente. En la figura 2.16, las barras azules corresponden a algoritmos basados en DL y la barra verde corresponde a métodos clásicos de clasificación de imágenes.

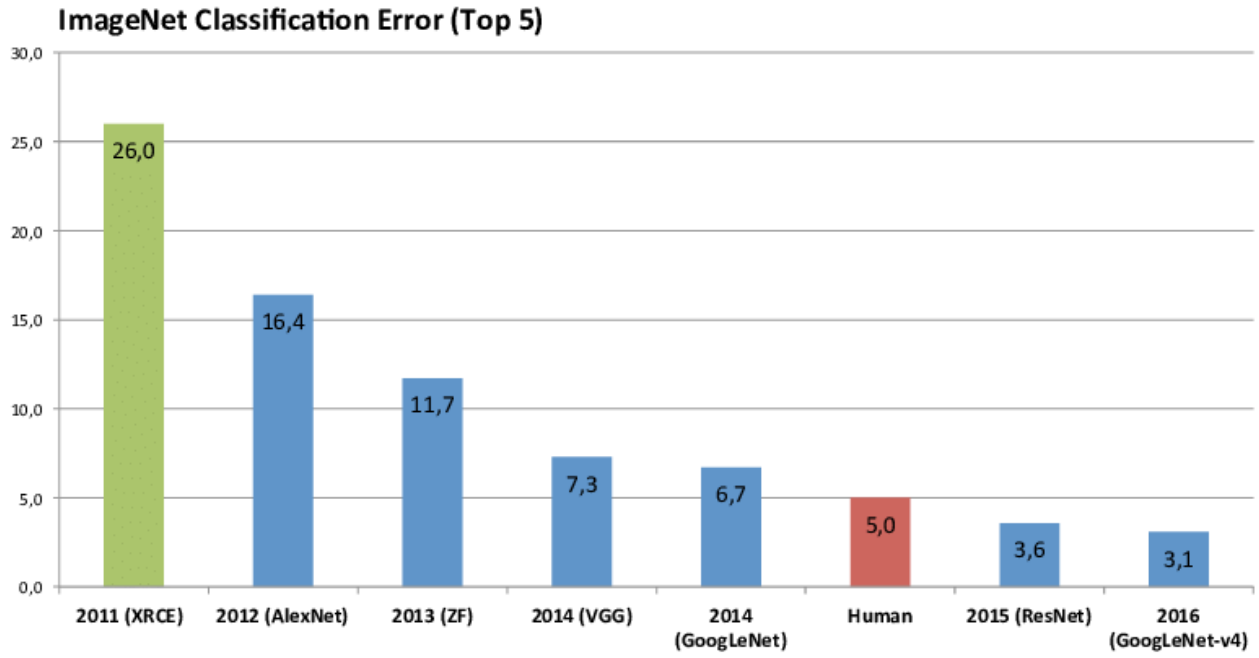


Figura 2.16: Resultados obtenidos en el dataset ImageNet entre 2011 y 2016.
Figura obtenida de: https://devopedia.org/imagenet.

A continuación, se hará una breve introducción a algunos de los conceptos más importantes de DL que se aplican en este trabajo. En particular, Redes Convolucionales y algunas arquitecturas conocidas, Redes Recurrentes, LSTM y *encoders*.

2.1.2.3.2. Redes Convolucionales (*CNN*)

Capa Convolucional

Estas redes realizan extracción de características de lo que se encuentre en el *input*. Realizan convoluciones y filtros para obtener estas características. En esta capa se ajustan los pesos de los filtros en el proceso de aprendizaje. Existen distintos tipos de capas convolucionales según la dimensión, la cantidad de neuronas y la función de activación que estas tengan.

La figura 2.17 muestra una capa convolucional 2D que aplica un filtro según la matriz *Kernel*:

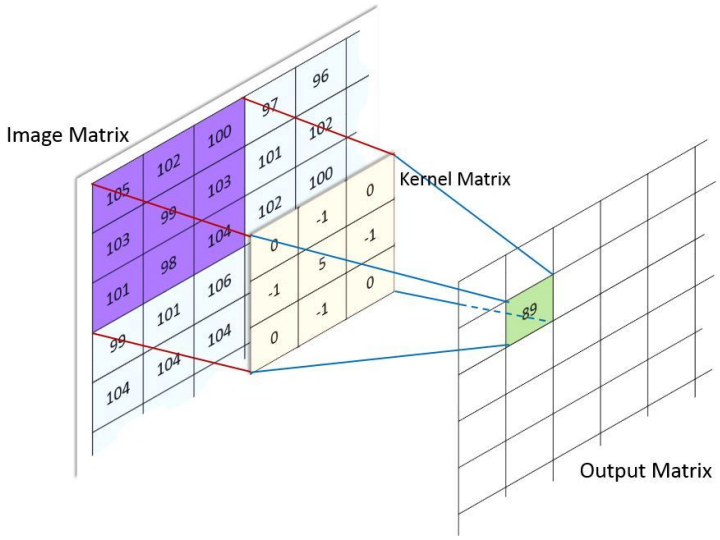


Figura 2.17: Ejemplo una capa convolucional.

Max Pooling

Max Pooling se utiliza para reducir las dimensiones de una imagen o matriz. Este método toma los máximos de las sub-matrices del arreglo original y crea, en base a esto, un nuevo arreglo. La figura 2.18 muestra un ejemplo para un núcleo de *max pooling* de 2×2:

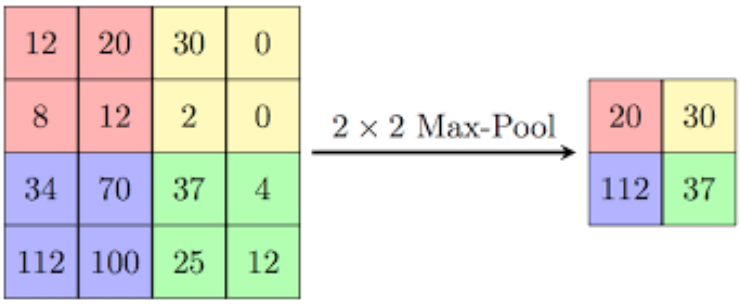


Figura 2.18: *Max pooling* de $2x2$. Imagen obtenida de https://computersciencewiki.org/index.php/File:MaxpoolSample2.png.

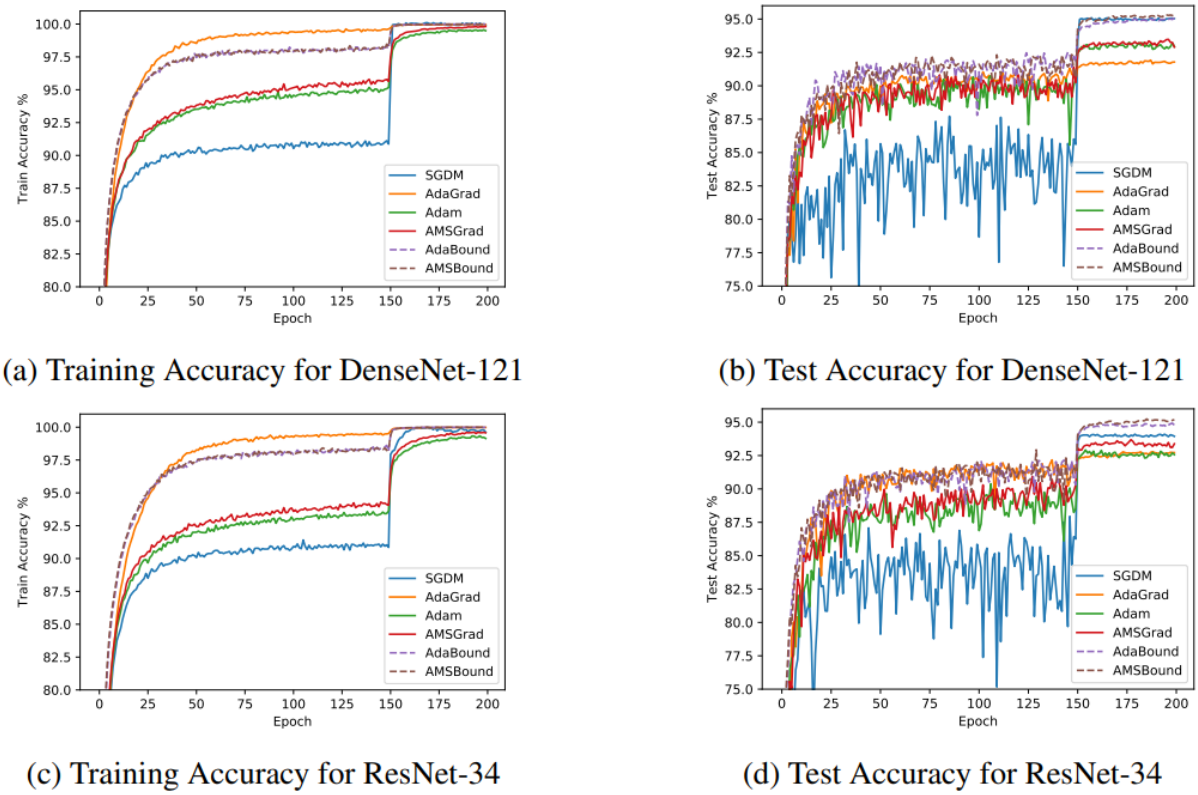
Optimizadores

En esta sección se revisará un optimizador particular utilizado en el proceso de entrenamiento de las redes neuronales convolucionales. Este se utilizó en la fase de exploración de modelos como alternativa al optimizador común usado *Stochastic gradient descent* (SGD).

AdaBound

AdaBound, presentado en Luo, Xiong, Liu, y Sun (2019), junta las ventajas de los optimizadores adaptativos, la velocidad de convergencia, con los métodos de gradiente estocásticos, la capacidad de generalización. La intuición detrás del método es que, inicialmente, el optimizador tenga características de los adaptativos (la velocidad) y que al final del entrenamiento mejore la capacidad de generalización.

La figura 2.19 muestra los resultados de la implementación de este optimizador en distintas arquitecturas y su efecto en la convergencia de *accuracy* tanto en entrenamiento como test:



(a) Training Accuracy for DenseNet-121

(b) Test Accuracy for DenseNet-121

(c) Training Accuracy for ResNet-34

(d) Test Accuracy for ResNet-34

Figura 2.19: *Accuracy* en entrenamiento y test para DenseNet-121 y ResNet-34 en el dataset CIFAR-10. En los gráficos, la línea morada corresponde a AdaBound. Imagen obtenida de (Luo et al., 2019).

2.1.2.3.3. Redes Neuronales Recurrentes (RNN) y LSTM

Las redes neuronales recurrentes son una arquitectura que se utiliza para el procesamiento de datos secuenciales, es decir, donde existe un concepto de orden entre los datos de entrada y salida. La razón por la cual se utilizan en este tipo de datos es porque existe una retroalimentación en la red con las predicciones previamente generadas. De esta forma, se genera cierta intuición de que estas redes tienen "memoria".

La figura 2.20 muestra una representación de cómo funcionaría una red neuronal recurrente con una secuencia de 4 datos:

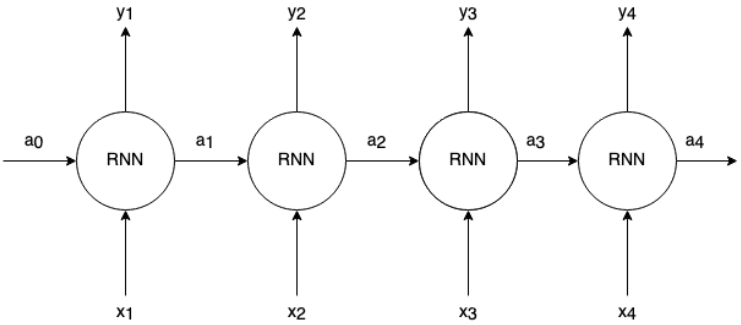


Figura 2.20: Esquema de procesamiento de secuencia de largo cuatro con una red neuronal recurrente.

En la figura 2.20, los datos de entrada son representados por $\tilde{\mathbf{x}} = \{x_1, x_2, x_3, x_4\}$, son alimentados a la misma red neuronal representada por los círculos. Es decir, cada elemento de la secuencia se entrega a la red y, el resultado de la activación o *hidden state* de la iteración también es alimentado a esta.

En particular, en este trabajo, se utilizan redes LSTM (*Long Short-Term Memory*) que corresponden a la familia de *Gated RNN*. Esta familia se origina debido a un problema en la propagación de los estados anteriores en la iteración actual de cada predicción. Dada la ecuación 2.4 que representa el estado oculto (*hidden state*) en el instante t:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \theta) \tag{2.4}$$

En la ecuación, $x^{(t)}$ representa la entrada actual, θ el conjunto de parámetros de la red neuronal, $h^{(t)}$ y $h^{(t-1)}$ los estados ocultos en t y $t-1$, respectivamente. Se observa que la dependencia de todos los estados anteriores se ve concentrada en $h^{(t-1)}$.

Las redes de la familia *Gated RNN* introducen compuertas (por eso el nombre *gated*) que evitan que al calcular el gradiente este diverja o se anule. Estas compuertas o "vías" auxiliares permiten que los estados anteriores no pierdan importancia en la medida que se obtenga el gradiente en el proceso de entrenamiento. Esto se debe evitar porque en el momento en que el gradiente se anula, la red pierde la capacidad de seguir aprendiendo. La figura 2.21 muestra una celda básica de las redes neuronales LSTM:

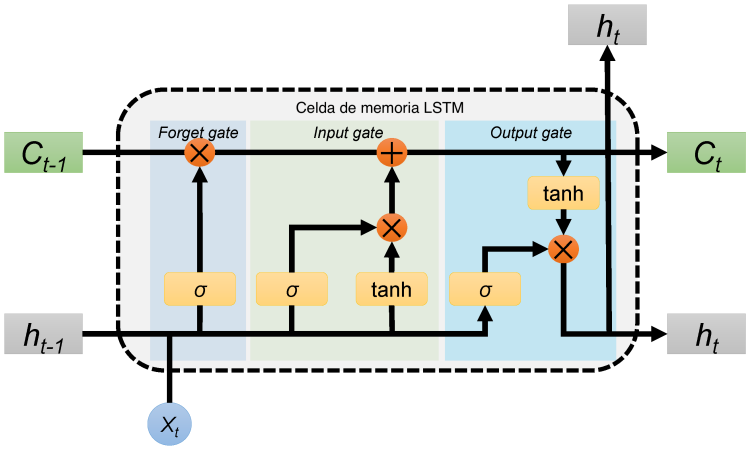


Figura 2.21: Celda de red LSTM. Figura obtenida de: https://www.mdpi.com/2073-4441/12/1/175/htm.

En la figura 2.21 se introduce el estado de la calda C_t. Esta variable permite que la información de los estados anteriores tomen más peso en cada paso por la red neuronal. Para comprender de mejor forma el comportamiento de las celdas LSTM, se describen a continuación las distintas compuertas o *gates*.

Forget Gate o compuerta de "olvido" determina mediante el producto matricial punto a punto la importancia de los estados de celda anteriores en la salida actual. Esto se observa en la ecuación 2.5:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.5}$$

La salida f_t es un vector al que se le ha aplicado la función sigmoidal, que entrega resultados entre $[0, 1]$ que indica cuanto valor se le da al estado de celda anterior. W_f y b_f corresponden a los parámetros entrenables de esta puerta. Seguido está la compuerta de entrada, *input gate* que decide que valores se utilizarán para actualizar el valor de la celda actual. La ecuación de esta compuerta en 2.6 sigue una forma similar a la *forget gate*:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_i] + b_i) \tag{2.6}$$

En la ecuación, la salida i_t es un vector al que se le aplicó la función sigmoidal, cuyos valores se encuentran entre $[0, 1]$. W_i y b_i corresponden a los parámetros entrenables.

Para la etapa final y salida de la celda, *output gate*, se deben tener en cuenta las siguientes ecuaciones:

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_i] + b_c), \tag{2.7}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.8}$$

Las ecuaciones 2.7 y 2.8 explican la actualización del estado de la celda C_t. W_c y b_c corresponden a los parámetros entrenables. *tanh* representa la función tangente hiperbólica. Finalmente, para el cálculo de la salida de la celda h_t, se tiene el siguiente par de ecuaciones:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_i] + b_o), \tag{2.9}$$
$$h_t = o_t * tanh(C_t) \tag{2.10}$$

El par de ecuaciones 2.9 y 2.10 corresponden al paso final para computar la salida de esta celda LSTM representada en la figura 2.21. En la ecuación 2.9, W_c y b_c corresponden a los parámetros entrenables. Las ecuaciones presentadas ayudan a comprender cómo las redes LSTM incluyen estados o resultados anteriores en las nuevas predicciones.

2.1.3. Métricas de evaluación del clasificador

Para la explicación de todas las métricas a continuación, incluida la matriz de confusión, se utilizará un mismo ejemplo para visualizar y realizar cálculos:

Supongamos se tiene un sistema que utiliza ML para discriminar el estado de salud de un paciente en base a algún examen [2], es decir, si tiene o no cáncer de pulmón en base a la radiografía cardiotorácica. Este modelo se ha entrenado con datos históricos de pacientes antiguos y se está poniendo en evaluación con un nuevo *set* de pacientes (conjunto de *test*).

Los resultados de las predicciones se muestran en la tabla 2.1:

Número de individuo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Clasificación Real	1	1	1	1	1	1	1	1	0	0	0	0
Predicción	0	0	1	1	1	1	1	1	1	0	0	0
Resultado	FN	FN	TP	TP	TP	TP	TP	TP	FP	TN	TN	TN

Tabla 2.1: Resultados obtenidos al predecir estado de pacientes. Se utilizan datos de https://en.wikipedia.org/wiki/Confusion_matrix

La fila de resultados se utilizará para la construcción de la matriz de confusión y para el cálculo de el resto de las métricas de evaluación del modelo. En adelante, se hablará indistintamente de la clase 1 como *positiva* y la clase 0 como *negativa*.

[2] Este problema ya ha sido enfrentado utilizando redes neuronales convolucionales como se plantea en Ronneberger, Fischer, y Brox (2015)

2.1.3.1. Matriz de confusión

La primera métrica de evaluación que se observa corresponde a la matriz de confusión. Esta se construye a partir del conteo de los siguientes resultados:

- **TP**, *True Positive*: son aquellas predicciones positivas que efectivamente eran positivas.
- **TN**, *True Negative*: predicciones negativas que eran negativas.

  Usualmente, se busca maximizar **TP** y **TN**, elementos que estarán siempre en la diagonal e indican correcto funcionamiento del modelo.

- **FP**, *False Positive*: Este error es cuando el modelo indica clase 1 o positivo (presencia de cáncer) cuando no había.
- **FN**, *False Negative*: Este error es cuando el modelo indica clase 0 o negativo (no presencia de cáncer) cuando había.
- **P**, representa el total de casos positivos.
- **N**, corresponde al total de casos negativos. $P + N$ es el total de de casos.

A partir de los resultados anteriores, se construye la matriz de confusión que se observa en la tabla 2.2:

		Predicción	
	Total $8 + 4 = 12$	**Cáncer**	**No-cáncer**
Condición real	**Cáncer**	6 (**TP**)	2 (**FN**)
	No-cáncer	1 (**FP**)	3 (**TN**)

Tabla 2.2: Matriz de confusión obtenida al realizar predicciones en conjunto de *test*.

Respecto a la matriz de confusión, siempre se intenta maximizar los valores que se encuentren en la diagonal. Dependiendo también de la aplicación, es factible buscar minimizar algún otro valor. En el ejemplo planteado del modelo detector de cáncer, lo lógico es evitar los **FN**, es decir, indicar que un paciente no tiene cáncer cuando si tiene.

A partir de los valores obtenidos en la matriz de confusión, se pueden obtener otras métricas de evaluación para el modelo que son comúnmente utilizadas en problemas de clasificación.

2.1.3.2. *Accuracy*

Esta métrica se define con la siguiente expresión matemática:

$$ACC = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.11}$$

En problemas y datasets donde las clases se encuentran balanceadas (y hay pocas o sólo dos clases), esta métrica es buen indicador. Si no es el caso, se deben utilizar otras que ayuden a describir mejor el tipo de error que se está cometiendo. Por ejemplo, si hay 90 positivos y 10 negativos, el clasificador podría predecir que todos los ejemplos son positivos y obtendría un accuracy de 0.9, lo que podría implicar un buen resultado en el entrenamiento del modelo. Pero si el interés de negocio es detectar los casos negativos, los resultados no son buenos.

2.1.3.3. *Precision*

Definida por la ecuación 2.12, es un indicador de cuantas de las predicciones positivas eran efectivamente positivas.

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2.12}$$

2.1.3.4. *Recall*

Corresponde a la cantidad de casos positivos que efectivamente se predijeron positivos.

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.13}$$

2.1.4. *F1 Score*

Se define usualmente como la media armónica entre *precision* y *recall*(ecuación 2.14). Se utiliza para combinar en un solo valor lo obtenido para estas métricas.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})} \tag{2.14}$$

2.2. Estado del arte

Debido a la naturaleza del problema (detección de comerciales en la radio), es difícil encontrar *papers* o programas que realicen lo mismo. Por otro lado, la radio como fuente de datos para clasificación de audio tampoco es algo que se encuentre en la literatura. Sí se trabaja el problema de clasificación de sonidos y distintas aplicaciones que procesan la voz y el sonido en general. Los principales problemas en el área corresponden a: clasificación de sonido (por ejemplo distinguir instrumentos), detección de momentos de silencio, eliminación de ruido, *speaker diarization*[3], *speech-to-text*.

Dado lo anterior, el estado del arte se basa en recopilación de *papers* y bibliografía útil en materia de extracción de características, modelos de clasificación de audio e imágenes.

2.2.1. Uso de espectrogramas y CNN

2.2.1.1. *CNN Architectures for large-scale audio classification*

Hershey et al. (2017) utilizó espectrogramas para extraer características y distintas arquitecturas de redes convolucionales profundas. La tarea en la que se enfoca este *paper* es en clasificar distintos sonidos con etiquetas como musica, ruido ambiental, conversación, sonido de auto, etc... en un *dataset* compuesto por cerca de 70 millones de videos de *youtube*.

En el *paper* se utilizan cuatro arquitecturas de redes neuronales como clasificador: *AlexNet*, introducida en Krizhevsky et al. (2012); VGG, *Inception V3* y *ResNet-50* (VGG, *Inception V3* y *ResNet* se explicarán en detalle en esta sección). Estas se comparan con una red *fully connected* que se implementa como modelo base (*baseline*). A continuación, se detalla brevemente tres de las arquitecturas mencionadas. Estas se utilizaron en el proceso de desarrollo del modelo de detección de radios.

2.2.1.1.1. VGG

Esta arquitectura fue propuesta por Simonyan y Zisserman (2015) y se implementó para resolver el problema de clasificación de imágenes ImageNet (Deng et al. (2009a)). El principal salto de esta arquitectura a diferencia de las contemporáneas corresponde a la profundidad de las capas ocultas. Otra diferencia es el tamaño de *kernel* de las capas convolucionales: usualmente se utilizaban filtros grandes ($11x11$) como campos receptivos. VGG en cambio implementa filtros de $3x3$ que corresponde a lo mínimo para que la entrada no sea una transformación lineal pixel a pixel sino que toma en cuenta información de lo que rodea a este para el cómputo de la salida de este filtro.

La figura 2.22 muestra un resumen de las arquitecturas propuestas en el *paper* original:

[3] *Speaker Diarization* corresponde a la tarea de separa en fragmentos homogéneos un fragmento de audio en base al hablante que se identifique.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 2.22: Tabla con distintas arquitecturas VGG propuestas. La usada en esta tesis corresponde a VGG16, o VGG D. Imagen obtenida de *paper* Simonyan y Zisserman (2015).

2.2.1.1.2. Inception V3

La arquitectura VGG entregó mejores resultados que su predecesora AlexNet (Krizhevsky et al. (2012)) en el problema ImageNet, pero aumentó considerablemente los costos computacionales de entrenamiento y predicciones debido a la necesidad de ir más profundo (*Very Deep Convolutional Networks* según como se presentó en el *paper* original). De hecho, tiene más del doble de parámetros ($62M$ *vs* $138M$) y en el proceso de entrenamiento se ejecutan hasta 13 veces más de operaciones por cada paso *forward* ($1,5$ *vs* $19,6$, en billones de operaciones de punto flotante, FLOP). Esto motiva a buscar una forma más eficiente de utilizar los recursos computacionales mediante la implementación de arquitecturas más livianas.

En Szegedy, Vanhoucke, Ioffe, Shlens, y Wojna (2015) se introduce Inception (en particular el *paper* corresponde a una reformulación de esta arquitectura), buscando reducir la cantidad de parámetros sin perder presición.

La principal innovación introducida por esta arquitectura corresponde al módulo *inception*. El propósito del modulo es remplazar las capas convolucionales que son computacionalmente caras, por ejemplo una convolución de $5x5$ por un conjunto de filtros mas pequeños con menos parámetros formando una arquitectura de red pequeña con menos parámetros. Un ejemplo de estos bloques *inception* se observa en la figura 2.23:

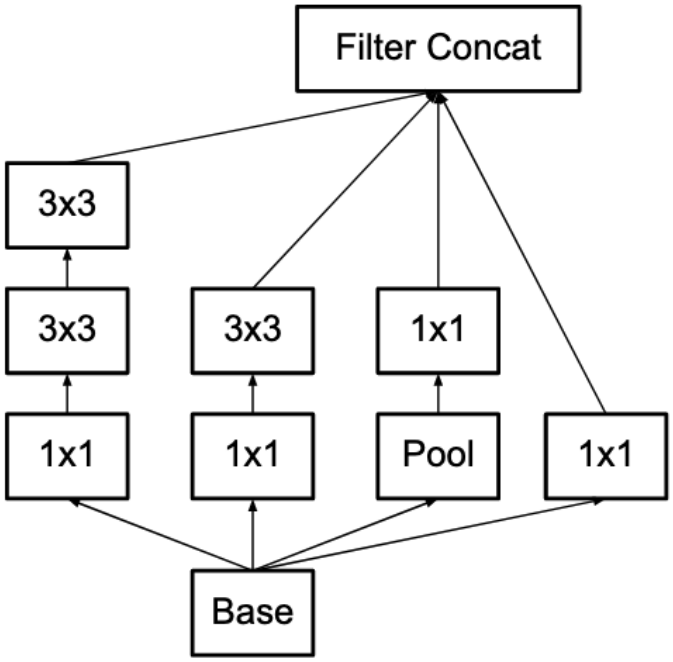


Figura 2.23: Bloque *Inception*. Imagen obtenida del *paper* Szegedy et al. (2015).

Otro cambio importante adoptado (no introducido) en esta arquitectura corresponde a los clasificadores auxiliares. Estos ayudan a "empujar gradiente" o a propagar mejor el error a las primeras capas de la red. Esto ayuda a la convergencia de la red combatiendo el problema del desvanecimiento del gradiente (también visto por las redes LSTM en la sección Redes Neuronales Recurrentes (RNN) y LSTM). La figura 2.24 muestra un ejemplo de red Inception en donde se observa el bloque de clasificación auxiliar:

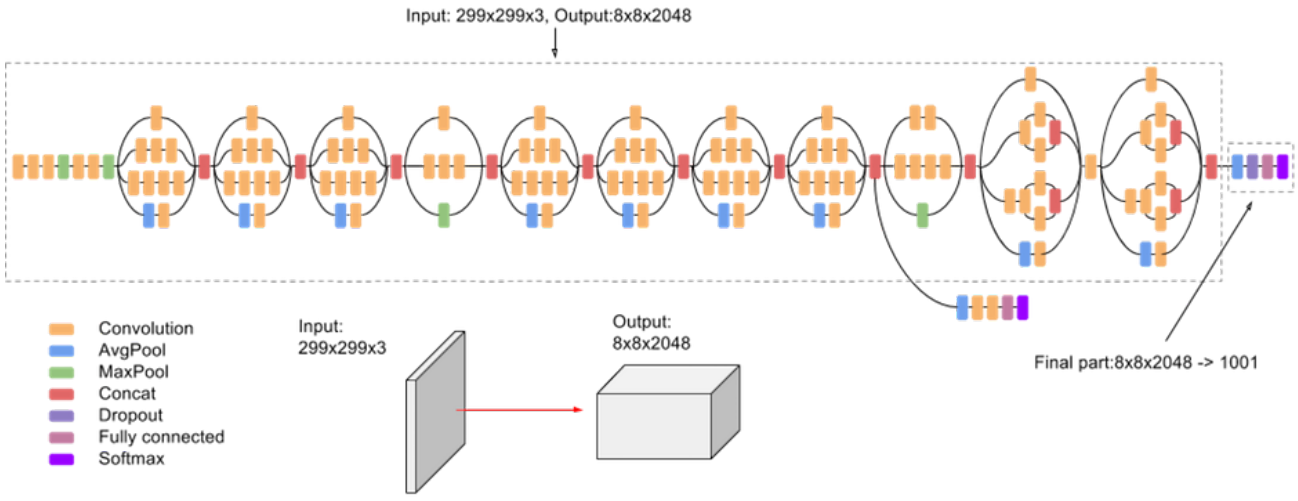


Figura 2.24: Ejemplo de implementación de Inception Net.

2.2.1.1.3. ResNet

En He, Zhang, Ren, y Sun (2015) se presenta una arquitectura *Deep Residual Learning* más profunda que las redes VGG (hasta 8 veces más profundas) pero menos compleja computacionalmente que la supera en la tarea de clasificación *ImageNet*, como *CIFAR-10*.

Con una motivación similar a lo propuesto en la arquitectura *Inception*, se propone ResNet para intentar resolver el problema de la convergencia o divergencia del gradiente en el entrenamiento al hacer las redes convolucionales más profundas. Para esto, se introduce una conexión directa entre capas llamado *residual learning block*. La figura 2.25 muestra cómo se construye este bloque:

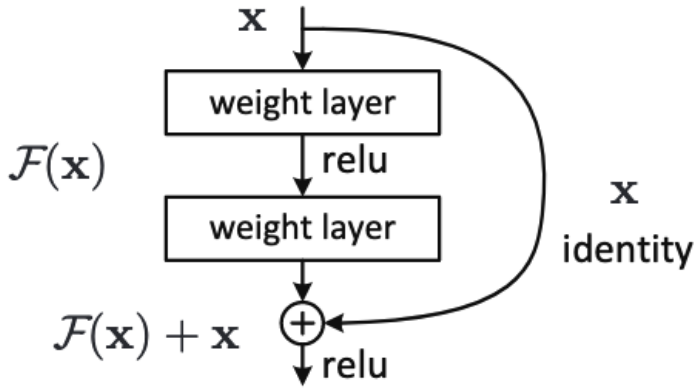


Figura 2.25: *Residual learning block*. Imagen obtenida del *paper* He et al. (2015).

Para *ImageNet* se proponen distintas arquitecturas que varían la cantidad de parámetros y el tamaño de los filtros. La figura 2.26 muestra las distintas combinaciones. En particular, en el trabajo mencionado (Hershey et al. (2017)) se utiliza ResNet-50 y en la implementación del clasificador de comerciales se utilizó la versión más liviana ResNet-18:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 2.26: Posibles arquitecturas de ResNet. Imagen obtenida del *paper* He et al. (2015).

A continuación, la tabla **??** resume los principales resultados obtenidos al utilizar las arquitecturas presentadas anteriormente para clasificar audio en base a la generación de imágenes a partir de los espectrogramas:

Arquitectura	Steps	Time	AUC	d-prime	mAP
Fully Connected	5M	35h	0.851	1.471	0.058
AlexNet	5M	82h	0.894	1.764	0.115
VGG	5M	184h	0.911	1.909	0.161
Inception V3	5M	137h	**0.918**	**1.969**	0.181
ResNet-50	5M	119h	0.916	1.952	**0.182**
ResNet-50	17M	356h	**0.926**	**2.041**	**0.212**

Tabla 2.3: Resultados obtenidos por las distintas arquitecturas propuestas en Hershey et al. (2017).

Palanisamy et al. (2020) utiliza distintas redes convolucionales profundas y compara el efecto de que estén (o no) pre-entrenadas. La tarea a resolver corresponde a ESC-50: Dataset for Environmental Sound Classification y otros problemas comunes de clasificación de audio. Algunas de las arquitecturas utilizadas: DenseNet, ResNet e Inception. En la tabla 2.4 se muestran algunos de los resultados obtenidos. Se muestra la comparación de desempeño de las distintas arquitecturas en base a si usaban o no el modelo preentrenado:

Modelo / *Dataset*	**GTZAN**		**ESC-50**		**UrbanSound8K**	
	Pre-entrenada	***Random***	**Pre-entrenada**	***Random***	**Pre-entrenada**	***Random***
ResNet	91.09 %	87.90 %	90.65 %	67.40 %	84.76 %	73.26 %
DenseNet	91.39 %	88.50 %	91.16 %	72.50 %	85.14 %	76.32 %
Inception	90.00 %	86.30 %	87.34 %	64.50 %	84.37 %	75.24 %

Tabla 2.4: Resultados obtenidos para distintas arquitecturas con distintas inicializaciones: preentrenadas (**ImageNet Deng et al. (2009b)**) o inicialización aleatoria.

2.2.2. Aplicaciones de *Deep Learning* para descriptores de audio

En esta sección se estudia en particular los resultados obtenidos en Wan, Wang, Papir, y Moreno (2020) (Investigación desarrollada en Google), donde se propone una nueva función de pérdida para el entrenamiento de modelos de *speaker verification*. Esta tarea consiste en identificar si cierto segmento de audio corresponde a un hablante determinado, basado en una comparación con una base de datos de hablantes conocidos.

Los autores del paper proponen una arquitectura de redes neuronales LSTM para la generación de *embeddings* (para efectos de este trabajo se hablará indistintamente de *encoder* y *embedding*) para describir audios que separan de buena forma los hablantes que son distintos. Para esto, realiza un tratamiento de ventana deslizante (similar a lo realizado en la figura 2.4) obteniendo coeficientes de MEL para cada ventana. Luego, estos se utilizan como entrada

en una red neuronal LSTM para obtener las predicciones que corresponden a vectores de características.

El *dataset* utilizado para el entrenamiento del *embedding* corresponde a uno creado por los investigadores en Google y está compuesto por 150 millones de grabaciones correspondientes a la frase "Ok Google" siendo mencionada por cerca de 630 mil hablantes distintos.

La figura 2.27 muestra como a partir de un segmento de audio se obtiene un *embedding*, es decir un vector de características o descriptor:

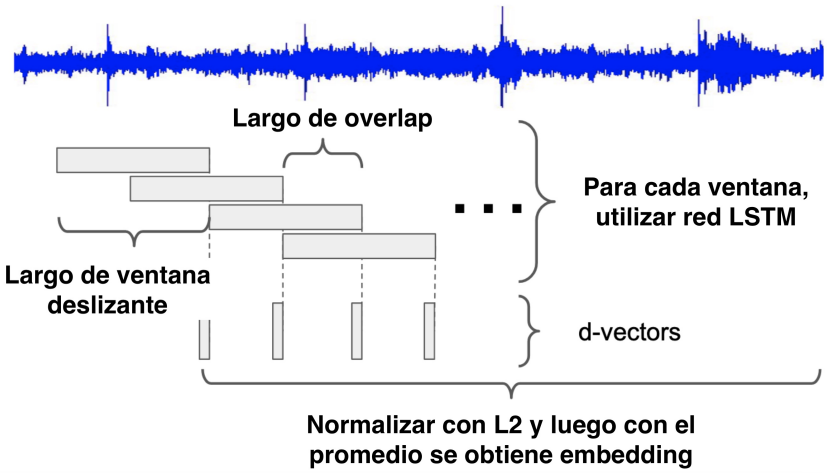


Figura 2.27: Uso de redes LSTM para obtención de *embeddings*.

Esta arquitectura se encuentra disponible en *Python 3* en una librería llamada **Resemblyzer**, la cual se utiliza en uno de los enfoques descritos más adelantes para la extracción de características. En particular, la capa de embedding o *voice encoder* es la que extrae caracteristicas para ser utilizadas en clasificadores.

Capítulo 3

Metodología y Datos

En esta sección se detallan las distintas herramientas utilizadas para el desarrollo de los modelos, creación del *dataset*, evaluación de resultados y técnicas utilizadas para mantener un desarrollo ordenado a lo largo del trabajo.

3.1. Herramientas utilizadas

La principal herramienta utilizada para el desarrollo de los modelos, procesamiento y preparación de los datos, gráficos y otros, corresponde a `Python`. Este lenguaje de programación se utilizó en conjunto con distintos *frameworks* y librerías que facilitan la implementación de los distintos módulos. A continuación, se listan los más importantes junto con una breve descripción:

- `Tensorflow` (Abadi et al., 2015), utilizada para la implementación de redes neuronales.
- `Pytorch` (Paszke et al., 2019), utilizada para la implementación de redes neuronales.
- `Scikit-learn` (Pedregosa et al., 2011) (o de forma equivalente, `sk-learn`, utilizado para modelos de aprendizaje basados en árboles y *Support Vector Machine*).
- `Librosa` (McFee et al., 2020), utilizada para el procesamiento de las señales de audio.
- `Resemblyzer` (Wan et al., 2020), utilizado para la extracción de características de audio.

Se utilizaron distintas herramientas para el desarrollo ordenado como *code linting*, y *GitHub* para el control de versiones de lo desarrollado.

Para el proceso de entrenamiento de los modelos se utilizó `Google Colab`. Esta plataforma permite la creación y ejecución de *jupyter notebooks* con la opción de uso de **GPU** de forma gratuita (uso limitado y no asegura disponibilidad).

3.2. Creación *Dataset*

Antes de comenzar con el desarrollo se necesita crear un *dataset* que permita entrenar distintos modelos. Para esto, se etiquetaron manualmente más de 20 horas de grabaciones de radio utilizando *Label-Studio* (Tkachenko, Malyuk, Shevchenko, Holmanyuk, y Liubimov, 2020-2021). La interfaz de *Label-studio* permite que no-programadores puedan etiquetar datos de audio de forma fácil haciendo que la creación del *dataset* sea más amigable.

Para etiquetar las 20 horas de audio, se dividen en segmentos pequeños entre 15$[s]$ a 60$[s]$, según detecciones de instantes de silencio en la señal de audio. Los instantes de silencio corresponden a una fracción de segundo en el que el volumen es 16$[dB]$ menor al volumen promedio del audio entero. Esto para facilitar el etiquetado haciendo que fuese una selección múltiple entre las posibles clases de cada muestra de audio. En la figura 3.1 se muestra la interfaz de usuario para el etiquetado:

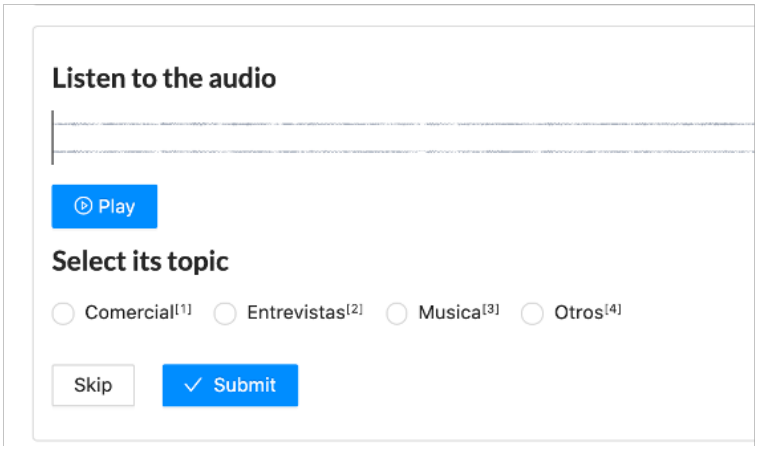


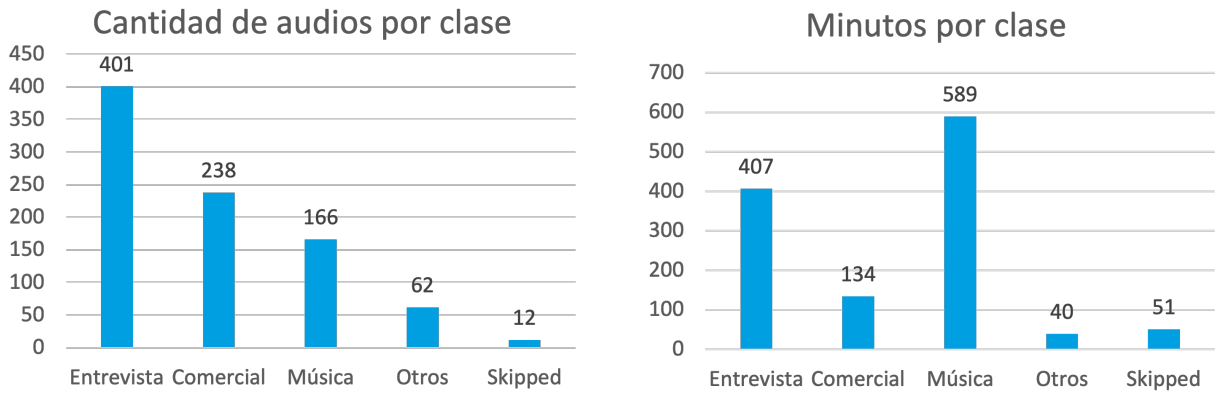
Figura 3.1: Interfaz creada con*label-studio*, utilizada para el etiquetado de los datos.

De esta forma, se generó un *dataset* con 1220$[min]$ de audio divididos en las siguientes clases:

- **Comercial:** segmento de comercial y propaganda. No considera auspicios mencionados por locutor o locutora.
- **Entrevista:** segmento del programa de radio. Puede ser llamada, entrevista o programación normal, etc...
- **Música:** segmento o canción completa.
- **Otros:** anuncios (realizados por grabaciones) sobre la programación o anuncio sobre el clima.
- ***Skipped:*** son aquellos en que no se pudo distinguir una categoría clara.

El motivo por el cual inicialmente se separó en 4 clases (5 si se incluyen aquellos audios que se saltaron en el etiquetado), es debido a que esto permite realizar un análisis exploratorio de herramientas para extracción de características de audio.

El criterio para decidir una categoría era cuando sobre el 95 % del segmento escuchado correspondía a alguna de las categorías. En caso contrario, se saltaba. En la figure 3.2 se observa un resumen de lo obtenido en el proceso de etiquetado:



(a) Cantidad de audios por clase.

(b) Minutos por clase.

Figura 3.2: Dataset obtenido.

Debido a que el planteamiento del problema original corresponde a implementar un detector de comerciales, el *dataset* se simplificó a uno de detección binaria con sólo dos clases: **Comercial** y **No Comercial**.

Es importante mencionar que la distribución del origen o radio emisora de cada segmento de audio no está balanceada, tanto en cantidad de muestras como en minutos de grabación por emisora. Esto afectó el trabajo dado que en los modelos desarrollados, se observaba desigual desempeño dependiendo de la emisora, en particular en producción. No se logró identificar una razón clara que explique este comportamiento. Una posible hipótesis corresponde a que los comerciales que se transmiten en distintas radios son distintos, lo que significa que menos comerciales para entrenar el modelo implican peor desempeños. La figura 3.3 muestra la cantidad de segmentos etiquetados por cada una de las fuentes utilizadas:

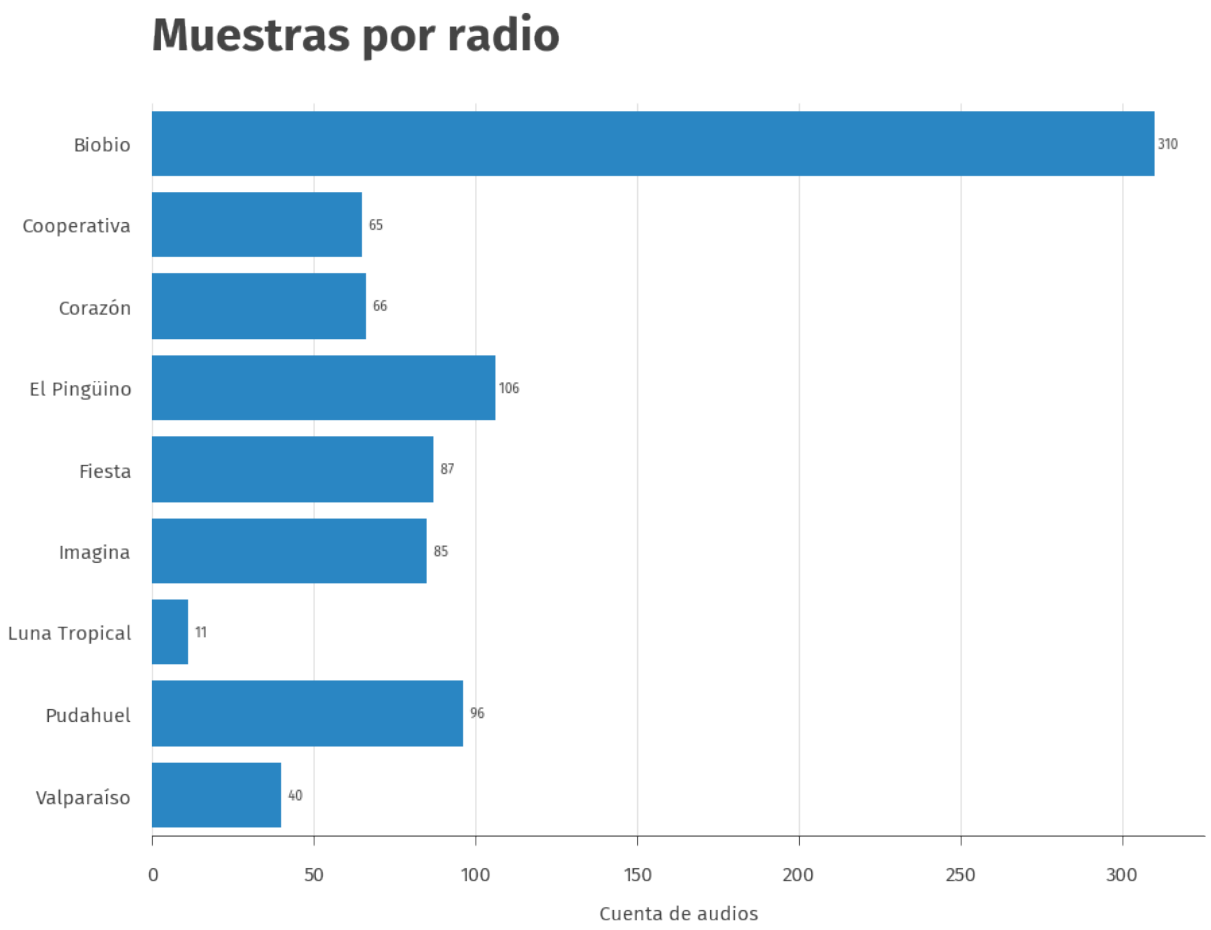


Figura 3.3: Cantidad de segmentos de grabación por radio emisora.

Para el proceso de entrenamiento y *test* se realiza una división del *dataset* con una proporción de 70 % para entrenamiento de los modelos y 30 % para *test*. Debido a la lentitud del proceso de etiquetado, a la falta de datos de entrenamiento y a que el modelo sería aplicado en producción, no se creó ni separó una fracción de los datos para validación. Se validará con los audios reales en entorno de producción.

3.3. Evaluación del modelo

3.3.1. Métricas de evaluación del clasificador

Para la evaluación de los modelos, dado que es un problema de clasificación binaria con un gran desbalance en las clases, se utilizará como principal métrica de evaluación la matriz de confusión, acompañada por otras métricas comunes como *accuracy*, *precision*, *recall* y *F1-score*, que fueron introducidas la sección 2.1.3.

3.3.2. Tiempo de ejecución

Debido a que el modelo resultante se utilizará en un entorno real (producción), es importante considerar una métrica para evaluar y comparar los tiempos de ejecución en el proceso de la implementación de los modelos. Esto permite comparar el desempeño de los estos no sólo pensando en métricas como *accuracy* si no en todos los aspectos de los ambientes de producción, como el tiempo de ejecución de este, siendo esto importante en el uso de recursos computacionales.

Como la forma actual de etiquetar o encontrar comerciales en la radio es de forma manual, se propone comparar el tiempo de procesamiento de un segmento con la duración de este. De esta forma, un humano escuchando un minuto de radio demoraría al menos un minuto en etiquetar. Se propone la siguiente fórmula para la tasa de procesamiento:

$$Tasa = \frac{\text{Tiempo de procesamiento}[s]}{\text{Largo de audio procesado}[s]} \tag{3.1}$$

Para la medición del tiempo de ejecución se utilizaron dos herramientas distintas:

- El decorador `%timeit` en *jupyter notebook*: esto permite medir el tiempo de ejecución de una celda de código ejecutándola cierta cantidad de veces y entregando un promedio de tiempo de uso de CPU.
- Libreria `time` de *Python*: al utilizar esta librería, era posible almacenar los valores de los tiempos de ejecución en arreglos de datos y luego graficarlos de forma más ordenada. Esto se utilizó para comparar librerías de extracción de características y procesamiento de audio.

3.4. Flujo general de trabajo

La figura 3.4 muestra como se llevó a cabo el proceso de extracción de datos, preparación, modelamiento, evaluación y puesta en producción:

Figura 3.4: Flujo de trabajo durante el desarrollo del modelo.

El flujo se considera desde la creación del *dataset* hasta la evaluación del modelo. Se utilizó esta metodología en el desarrollo ya que, una vez creado el *dataset*, permite la rápida iteración entre la implementación de un modelo y la evaluación. Este flujo no es tan distinto al desarrollo de proyectos en situaciones reales trabajando con clientes.

Capítulo 4

Implementación

4.1. *Pipeline* de procesamiento

A lo largo del proyecto, se trabajó con una misma línea de trabajo o *pipeline*. Esto permite realizar cambios en distintas etapas pero mantener una estructura general para mayor facilidad y rápida iteración. Los distintos procesos se separaban de forma que la tarea de cada uno fuese clara, de una sola responsabilidad y que se puedan crear distintas combinaciones a partir de cada bloque. La figura 4.1 ilustra de forma general el *pipeline* por el que pasa una señal de audio para obtener una clasificación:

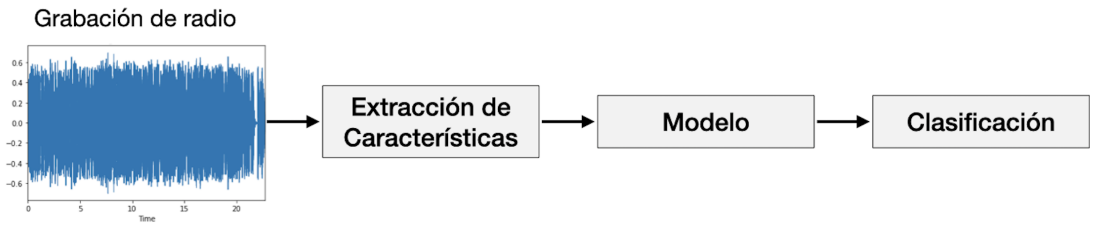


Figura 4.1: *Pipeline* propuesto como estructura de procesamiento de las señales.

Lo anterior, corresponde a una abstracción de lo desarrollado en donde cada uno de los bloques del proceso: extracción de características, entrenamiento y modelo y clasificación, se implementó con herramientas distintas. A continuación, se detallan distintas variaciones del *pipline* o enfoques implementados.

4.1.1. Enfoque 1: Espectrogramas de MEL y CNN

El procesamiento de audio en este enfoque, inspirado principalmente en (Palanisamy et al., 2020), (Collobert et al., 2016), (Hershey et al., 2017) y (Guzhov et al., 2020), consiste en transformar la señal de audio en una imagen y utilizar modelos de redes neuronales convolucionales como clasificador. La figura 4.2 ilustra de forma general como funciona este enfoque:

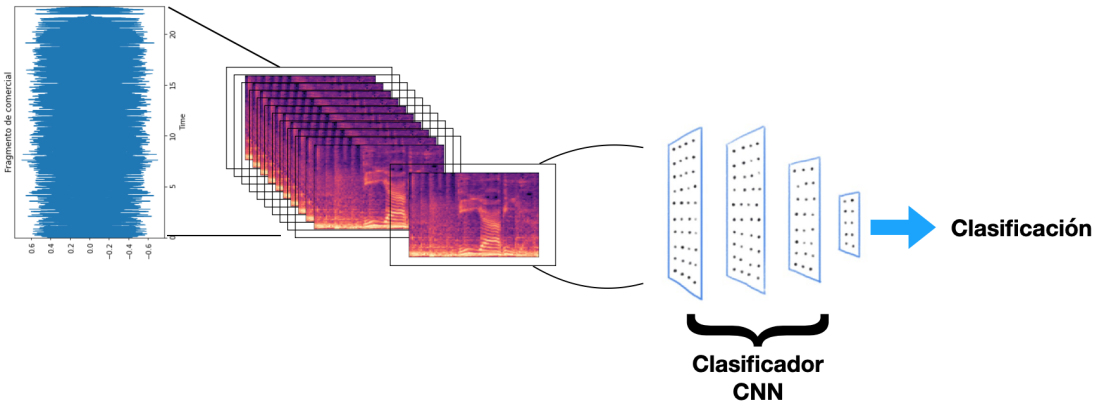


Figura 4.2: *Pipeline* enfoque 1: espectrogramas de MEL y CNN.

El proceso de extracción de características corresponde en hacer un análisis de ventana sobre los segmentos de audio y transformar los espectrogramas de MEL resultantes a imágenes, para ser almacenadas en disco duro. Al realizar este procedimiento, debido a que el procesamiento es utilizando ventanas de tiempo más pequeñas que la duración de cada audio, se obtienen más imágenes que segmentos de audio en el *dataset*. En particular, en esta implementación se usaron ventanas de $2[s]$ de ancho.

Para la generación de estas imágenes se probaron distintas librerías de procesamiento de señales o *frameworks* de `Python` ya mencionados: `Pytorch` (en particular el módulo presentado por (Yang et al., 2021)), `librosa` y `scipy` (Virtanen et al., 2020).

El proceso implementado para transformar los audios en imágenes es lento. Este proceso tomó alrededor de 26 días en convertir las 20 hrs de audio que comprendían el *dataset* a imágenes.

Este set de datos se utiliza posteriormente para entrenar un modelo de redes neuronales artificiales, en particular la arquitectura `ResNet-18`. La elección se basa en distintos criterios: de las arquitecturas presentadas en el *paper* citado[4] es la que tiene menor cantidad de parámetros entrenables: $11.7M$ versus $25M$ para `Inception V3` y $138M$ para `VGG16` (datos sobre cantidad de parámetros obtenidos de (Papers With Code, 2022)). Esto implica menor cantidad de recursos computacionales (en particular memoria RAM) al momento de entrenar el modelo y el uso de tamaños de *batch* más grande.

Aún dadas las limitaciones en recursos, se realizaron experimentos con arquitecturas dis-

[4] (Guzhov et al., 2020)

tintas. Se utilizó `VGG16` y `ResNet-18` con un cambio en optimizador: se implementó un algoritmo de regularización de decaimiento de pesos o `AdaBoundW`, propuesto en (Loshchilov y Hutter, 2017). Debido a los resultados obtenidos (ver imagen 5.1), no se sigue desarrollando este enfoque. Se implementaron también otros modelos con peores resultados que no se exponen en este informe: `VGG-16` presentada en (Simonyan y Zisserman, 2015), `Inception` introducida en (Szegedy et al., 2014) y otros modelos derivados de estas redes, realizando también cambios en la función de optimización.

4.1.2. Enfoque 2: Coeficientes de MEL y modelos de clasificación

Este enfoque surge a partir de la necesidad de procesar audios más rápidamente (al menos mejorar una tasa de 1 [5]) con el menor uso de recursos computacionales posible. Es decir, bajo consumo de CPU como RAM. A partir de esas necesidades, se decide utilizar modelos de rápida implementación en `sk-learn`. Para esto, se utilizó el mismo paradigma de procesar el audio utilizando ventanas temporales pero se transforma la información en vectores de largo n, dependiendo de la cantidad de coeficientes de MEL que se requieran utilizar. La figura 4.3 muestra cómo se adapta el *pipeline* a este enfoque:

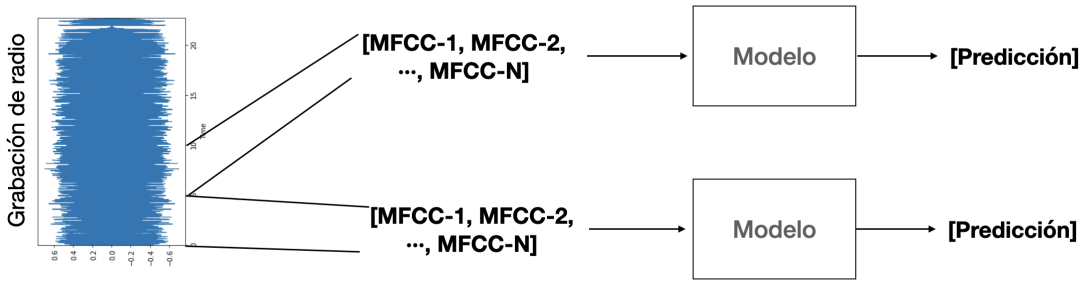


Figura 4.3: *Pipeline* considerando un vector de características para ventanas de tiempo y modelo de `sk-learn`.

La principal ventaja de este modelo es que permite la rápida iteración entre entrenamiento y *test*. Se probaron modelos de clasificación basados en árboles como DT, RF y modelos SVM.

También, se prueban distintas configuraciones de ancho de ventana para las características y se varía la cantidad de coeficientes de MEL que componen un vector el características.

A continuación, la tabla 4.1 muestra las principales configuraciones entrenadas y probadas utilizando una búsqueda de grilla:

[5] Tomando en cuenta la definición de tasa en la ecuación 3.1

	Ancho de Ventana [s]	Coeficientes de MEL	Modelo de clasificación
1	2 [s]	40	Random Forest
2	2 [s]	40	XGBoost
3	5 [s]	40	XGBoost
4	5 [s]	40	Random Forest
5	5 [s]	60	Random Forest
6	12 [s]	40	Random Forest

Tabla 4.1: Configuraciones probadas utilizando en el enfoque 2.

Dado también que el origen de las grabaciones en el *dataset* no estaba igualmente distribuido, según se muestra en la figura 3.3, se propone en este enfoque una variación que implicaba entrenar un modelo para cada radio. Es importante mencionar que la distribución de las radio emisoras para el etiquetado inicial fue completamente aleatorio, sin preferir una emisora por sobre otra.

Esta variación no cumple con el objetivo específico que indica que el modelo debe detectar comerciales sólo utilizando información de audio, es decir sin datos sobre el origen u información adicional sobre este.

4.1.3. Enfoque 3: Resemblyzer y modelos de clasificación

El uso de `Resemblyzer` como extractor de características permite implementar el modelo clasificador con cualquier tipo de algoritmo de inteligencia artificial. Al igual que el enfoque anterior, el resultado de la extracción de características es un vector (esta vez de dimensión fija: $n = 256$) que puede ser procesado por modelos simples como los implementados con `sk-learn` (SVM o *Random Forest Classifier*) o redes neuronales.

La figura 4.4 muestra como a partir de una señal de audio, se puede obtener un vector de características que separe lo más posible clases distintas:

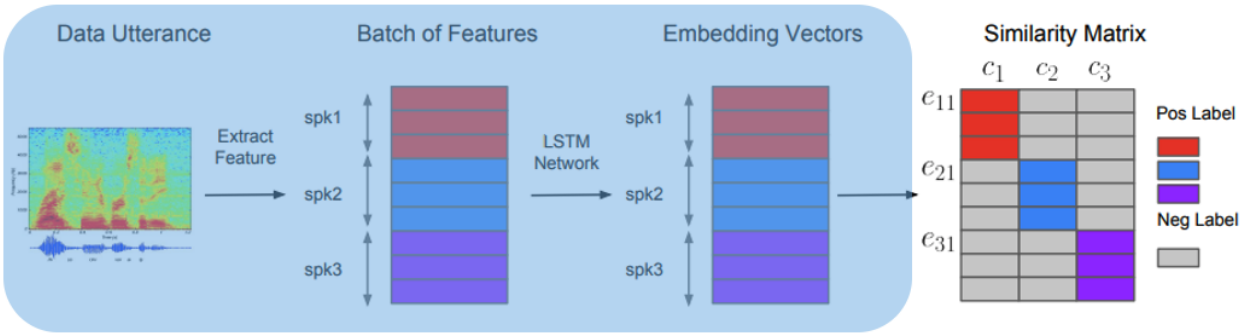


Figura 4.4: Uso de Resemblyzer (Wan et al., 2020) como extractor de características. Figura obtenida de (Wan et al., 2020).

Esto es resultado de la aplicación de lo descrito en la sección 2.2.2, en donde se muestra en la figura 2.27 cómo utilizando tratamiento de ventanas más redes LSTM, se obtienen vectores de características para un segmento de audio.

Utilizando esta metodología se implementaron distintos modelos: SVM, Random Forest y MLP de las librerías `sk-learn` y una red neuronal simple implementada en `Tensorflow`. En la tabla 4.2 se muestra la arquitectura de la red. Esta se eligió de esta forma debido a que el proceso de obtención de embeddings con *Resemblyzer* es complejo computacionalmente lo que implica largos tiempos de procesamiento. Para evitar aumentar considerablemente el tiempo de ejecución y entrenamiento, se implementó un clasificador con pocos parámetros:

Capa (tipo)	**Forma de salida**	**Param #**
Dense	(None, 256)	65792
Dropout	(None, 256)	0
Dense	(None, 128)	32896
Dropout	(None, 128)	0
Dense	(None, 128)	16512
Dropout	(None, 128)	0
Dense	(None, 2)	258
	Total de parámetros: 115,458	

Tabla 4.2: Arquitectura usada para clasificación en enfoque 3.

Capítulo 5

Resultados y Análisis

5.1. Resultados

A continuación se presentan los resultados obtenidos para cada enfoque presentado en la sección anterior.

5.1.1. Enfoque 1: Espectrogramas de MEL y CNN

Resnet-18 con optimizador `SDG`:

En la tabla 5.1 se observan los resultados obtenidos utilizando espectrogramas de MEL como entrada a la red `ResNet-18`:

		Predicción	
		Comercial	No-Comercial
Etiqueta Real	Comercial	85	14
	No-Comercial	13	235

Tabla 5.1: Mejor resultado obtenido a partir del uso espectrogramas de MEL y `ResNet-18`. *Accuracy* obtenida es 92.2 %.

Es importante destacar que a pesar de ser este el mejor resultado utilizando redes neuronales e imágenes como características, el tiempo de procesamiento es muy alto en particular en el proceso de extracción de características. La tasa (ver ecuación 3.1) según se define en la metodología, es del orden de $tasa \sim 32.2$. Dicho de otra forma, se demoró cerca de 3 días en procesar 134 minutos de audio.

Resnet-18 con optimizador `AdaBoundW`:

La figura 5.1 contiene la matriz de confusión de la implementación del enfoque 1 con la variante en la función de optimización, utilizando `AdaBoundW`. Como se comentó en la sección 4.1.1 (Enfoque 1: Espectrogramas de MEL y CNN), los resultados obtenidos muestran que solo se predijo en la clase mayoritaria que corresponde a música. Esto para las aplicaciones y puesta en producción, es el de los peores escenarios que se podría obtener debido a que, independiente de la cantidad de predicciones que se realicen, nunca se encontrarían los comerciales buscados.

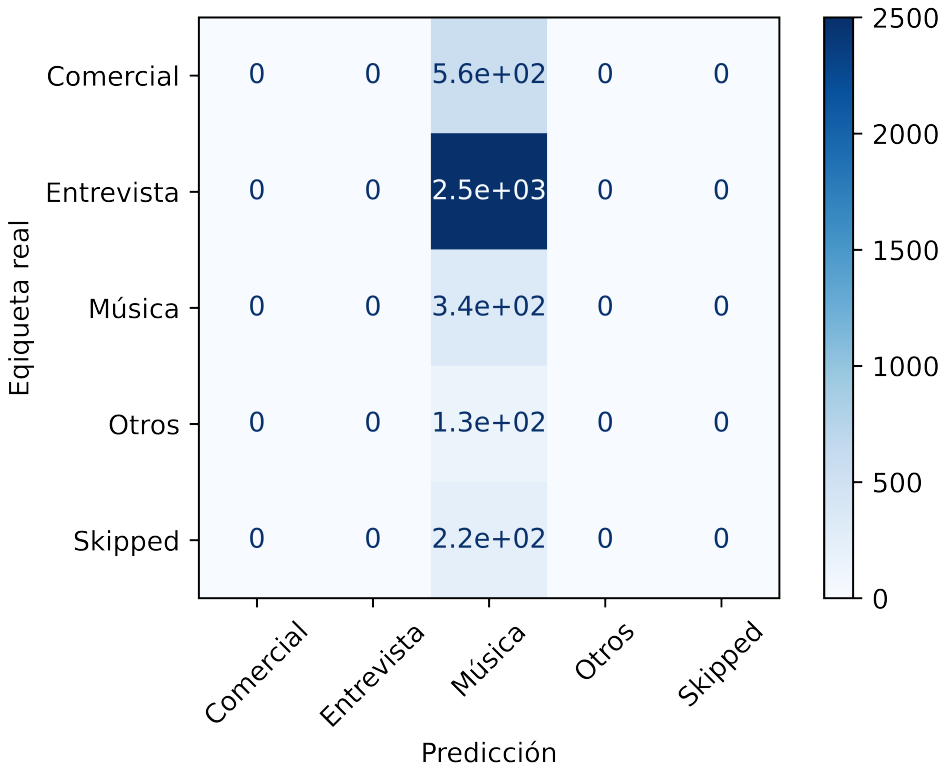


Figura 5.1: Resultados de ResNet-18 con AdaBoundW como optimizador.

Con respecto a los resultados en la figura 5.1, se observa que el orden de magnitud de la cantidad de predicciones realizadas es mucho mayor a lo que se observa en la tabla 5.1. Esto debido a que para los experimentos realizados con `AdaBoundW`, se realizaron predicciones no a nivel de audio sino, a nivel de ventana de $2[s]$ de características extraídas.

En este experimento se varía la forma en la que se generan las predicciones: se utiliza cada ventana para realizar predicciones y, luego, se agrupan por segmento de audio calculando la moda. De esta forma se implementó un "ensamble". Este último paso solo se implementó para la detección binaria.

En la tabla 5.2, se muestran los resultados de transformar la detección multiclase con `AdaBoundW` a un problema de clasificación binaria:

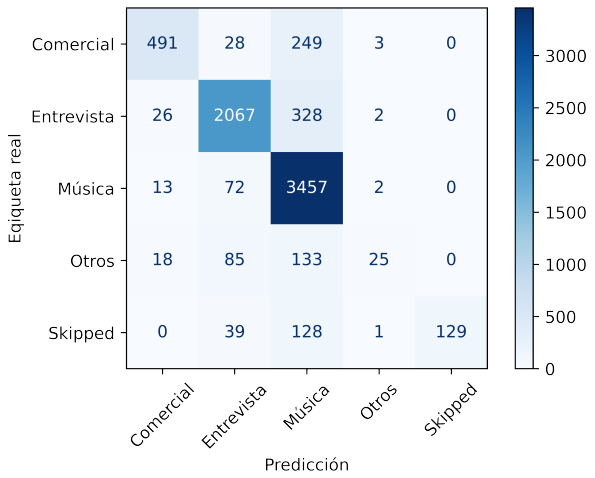
		Predicción	
		Comercial	No-Comercial
Etiqueta Real	Comercial	0	560
	No-Comercial	0	3190

Tabla 5.2: Resultados obtenidos con espectrogramas de MEL, `ResNet-18` y `AdaBoundW` como optimizador.

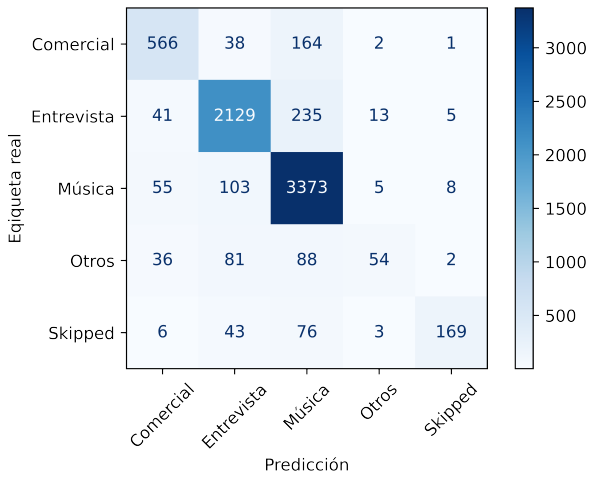
Los resultados obtenidos en 5.2 son consistentes con lo observado en la matriz de confusión en 5.1. Considerando que la predicción *Música* es consistente con *No Comercial*, entonces se espera el comportamiento obtenido. Lo interesante del experimento, es que el modelo obtiene los mismos resultados para tratamientos distintos de los audios; la clasificación multiclase se realiza sobre ventanas de 2$[s]$ y la detección binaria sobre segmentos completos.

5.1.2. Enfoque 2: Coeficientes de MEL y modelos de `sk-learn`

Las figuras 5.2, 5.3 y 5.4 muestra los resultados obtenidos por las distintas configuraciones de la tabla 4.1:

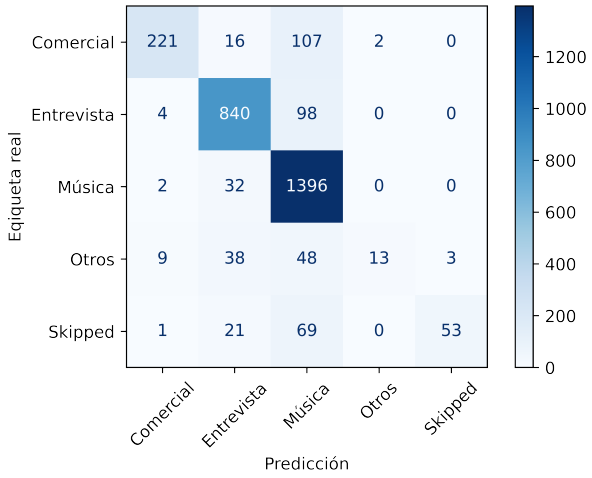


(a) Ventana de 2s, 40 MELs. RF como clasificador. *Accuracy* 84.6 %

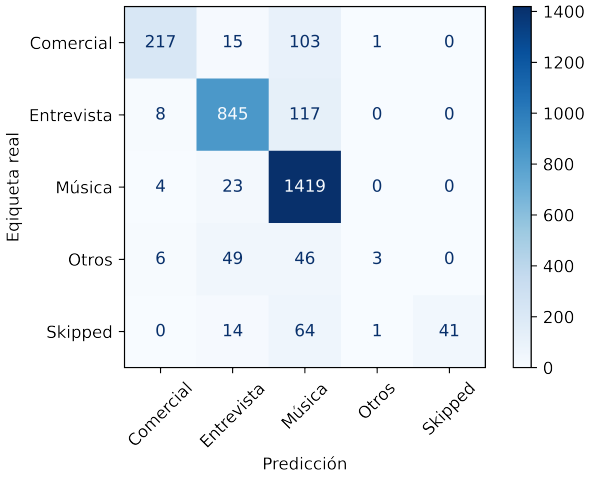


(b) Ventana de 2s, 40 MELs. XGBoost como clasificador. *Accuracy* 86.2 %

Figura 5.2: Resultados obtenidos por las distintas configuraciones.

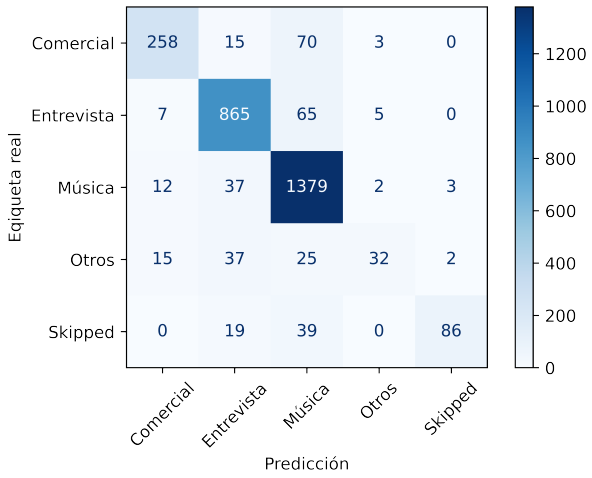


(a) Ventana de 5s, 40 MELs. RF como clasificador. *Accuracy* 84.9 %

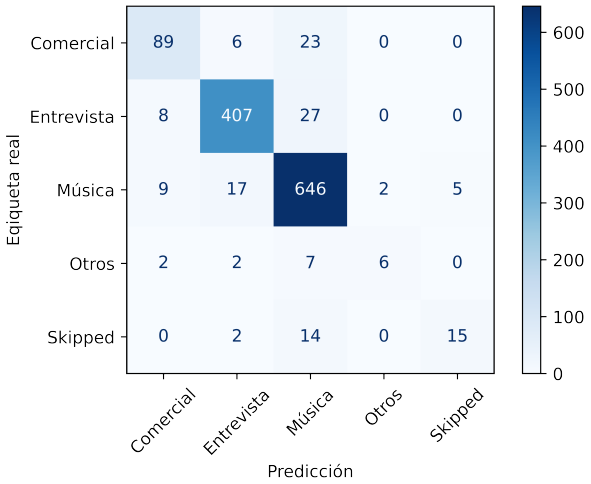


(b) Ventana de 5s, 60 MELs. RF como clasificador. *Accuracy* 84.8 %

Figura 5.3: Resultados obtenidos por las distintas configuraciones.



(a) Ventana de 5s, 40 MELs. XGBoost como clasificador.*Accuracy* 88.0 %



(b) Ventana de 12s, 40 MELs. RF como clasificador. *Accuracy* 90.4 %

Figura 5.4: Resultados obtenidos por las distintas configuraciones.

La tasa [6] obtenida con este enfoque es 0.04.

5.1.3. Enfoque 3: Resemblyzer

Los resultados obtenidos al entrenar una red neuronal pequeña (arquitectura en la tabla 4.2) con Resemblyzer como extracción de características se muestran en la tabla 5.3, a continuación:

[6] Definida en 3.1.

		Predicción	
		Comercial	No-Comercial
Etiqueta Real	Comercial	80	19
	No-Comercial	16	232

Tabla 5.3: Mejor resultado obtenido a partir del uso de Resemblyzer y modelos implementados con `sk-learn`.

La tasa de procesamiento para este método corresponde a 0.126, lo que es 3.15 veces más lento que el enfoque anteriormente desarrollado.

5.2. Puesta en producción

Previo al detalle de la puesta en marcha, se debe recordar el contexto en el que se desarrolla el detector de comerciales: El objetivo final (impuesto no en la memoria sino por la empresa) corresponde a facilitar y acelerar la búsqueda de nuevos comerciales y publicidades políticas en el contexto de campañas de elecciones.

Dado lo anterior, para la puesta en producción se utilizó el modelo obtenido con el enfoque 3: `Resemblyzer` con modelos de clasificación implementados con `sk-learn`. La elección se basa en dos criterios: una comparación de el tiempo de ejecución de cada modelo y el desempeño obtenido en la detección de comerciales. La infraestructura utilizada y proceso desarrollado para la grabación de radios y generación de predicciones se muestran en la figura 5.5:

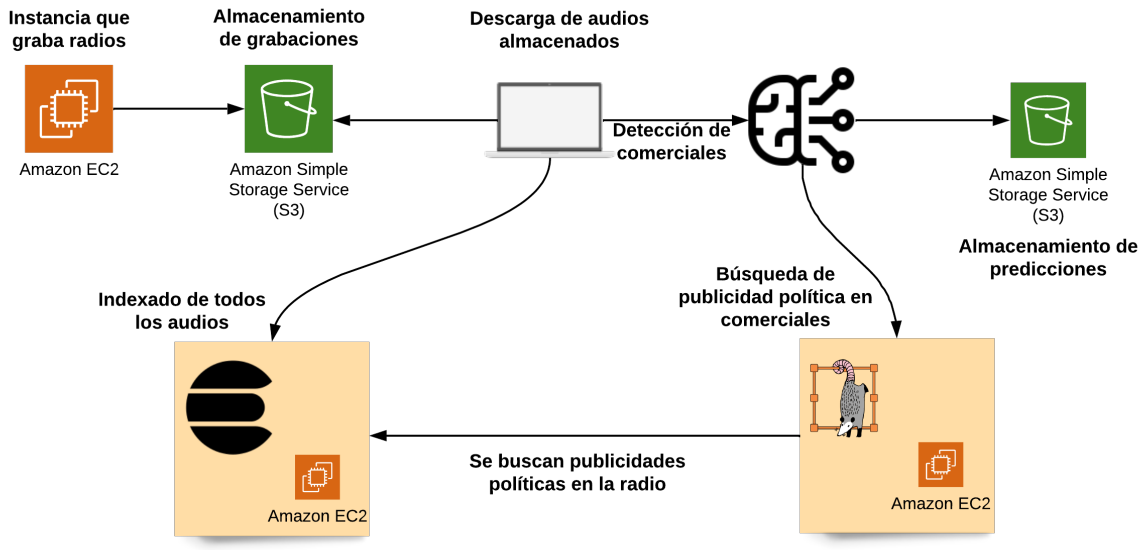


Figura 5.5: Infraestructura utilizada y proceso para grabación de radios y detección de comerciales.

Los dos primeros elementos corresponden al sistema que graba radios en ciertos horarios definidos por la empresa. Básicamente, se configura un archivo con las horas (usualmente entre 6am hasta 8pm) y radios a grabar, y un script de *Python* se encarga de grabar y almacenar en la nube con `S3` (*Simple Storage Service*, servicio de almacenamiento de AWS). Luego, en la medida que se le solicite (*on demand*), desde un computador se descargaban los audios para ser procesados.

Para poder indexar los audios, se extraen características únicas de las grabaciones, *fingerprints*. Estas huellas corresponden a una combinación de características que permiten identificar (en teoría) únicamente cada segmento distinto de sonido. Para este proceso, se utiliza `Chromaprint` [7].

Luego, se deben indexar todas las grabaciones (fingerprints) en *elasticsearch*. Elasticsearh es un motor de búsquedas y análisis distribuido de distintos tipos de datos basado en Apache Lucene[8]. En particular, en esta memoria se utilizan índices de *elasticsearch*, que corresponden a una colección de documentos del tipo JSON. La estructura de datos que se utiliza corresponde a Inverted Index. A medida que se agregan documentos (en este trabajo, las *fingerprints* de cada grabación) se crea un índice invertido que permite realizar búsquedas rápidas. *Elasticsearch* cuenta con una interfaz o API HTTP. Esto permite realizar rápidamente las búsquedas de las publicidades políticas encontradas.

Una vez indexados los audios se procede a la detección de comerciales. Esto ocurre en el mismo computador que descarga los audios, con grandes volúmenes de datos al mismo tiempo. Eso es posible debido a que el modelo utilizado es lo suficientemente eficiente como para procesar por completo una hora de grabación en menos de 3$[min]$. Una vez encontrados los posibles comerciales, estos segmentos (en teoría comerciales) se etiquetaban manualmente utilizando *label studio*, buscando comerciales políticos. En esta etapa, se utilizaron las siguientes etiquetas:

- **Comercial:** segmento correctamente detectado como comercial pero que no corresponde a publicidad relacionado a política.
- **No Comercial:** segmento incorrectamente detectado como comercial.
- **Político:** corresponden a los segmentos de comercial de publicidad política. Adicionalmente, se intentaba anotar a que partido o candidato/a pertenecía. Esta es una detección correcta.

Al finalizar la búsqueda de comerciales con el modelo y validada (de forma manual) esta selección, se procedía a realizar búsquedas de las publicidades encontradas en el índice de *elasticsearch*. Este proceso final genera los reportes y gráficos que se observan en la figura 1.1.

Este proceso de etiquetado cumplía un doble propósito: la búsqueda de nuevas publicidades políticas y la evaluación del *pipeline* implementado y puesta en producción. La figura 5.6 muestra los resultados obtenidos con el primer modelo implementado en producción.

[7] https://acoustid.org/chromaprint
[8] https://www.elastic.co/what-is/elasticsearch

Primera ronda de evaluación.

Para esta ronda de evaluación, se utilizaron audios de la radio Bíobio correspondientes a 3 hrs y 20 min. Se evaluaron segmentos que el modelo clasificó tanto como comercial y no comercial.

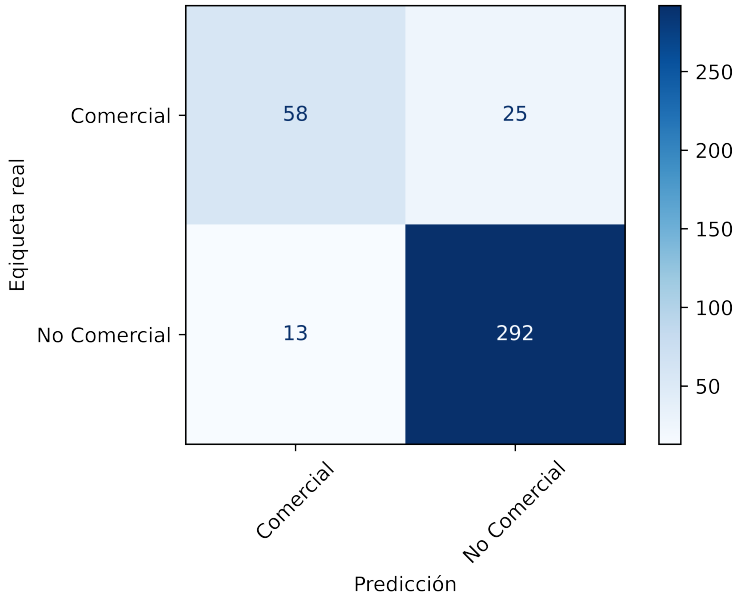


Figura 5.6: Resultados primera evaluación. Corresponden únicamente a la radio Bíobio. No se distingue la clase **Político** debido a que se une con la clase **Comercial**.

Los resultados obtenidos en esta primera ronda de evaluación permitieron encontrar en menor tiempo publicidades políticas (objetivo de la implementación del modelo). Para complementar el análisis es importante recurrir a métricas adicionales. La tabla 5.4 muestra en particular, *precision, recall* y *f1-score* respecto de la clase *comercial* y *no comercial*:

Clase	Precision	Recall	F1-score	Cantidad Muestras
Comercial	0.82	0.70	0.75	83
No Comercial	0.92	0.96	0.94	305

Tabla 5.4: Resultados obtenidos en producción en primera fase de evaluación. *Accuracy* obtenida es 90 %.

Segunda ronda de evaluación.

Se realizó una segunda ronda de validación de resultados utilizando *label studio*. En esta ocasión, los audios evaluados corresponden sólo a aquellos seleccionados como comercial por el modelo. Esto debido a que se quería agilizar la búsqueda de publicidad política en la radio. Debido a que la clase predicha es siempre comercial, hay algunas de las métricas especificadas

anteriormente (la precisión y F1-score respecto a la clase no comercial), no tienen sentido que en la tabla 5.5 quedan marcados como **-**:

Clase	Precision	Recall	F1-score	Cantidad Muestras
Comercial	0.41	1.00	0.58	145
No Comercial	-	0.00	-	211

Tabla 5.5: Resultados obtenidos en producción en segunda fase de evaluación.*Accuracy* obtenida es 41 %.

En esta ronda de evaluación, las grabaciones se distribuyen por radio emisora según lo que muestra la figura 5.7:

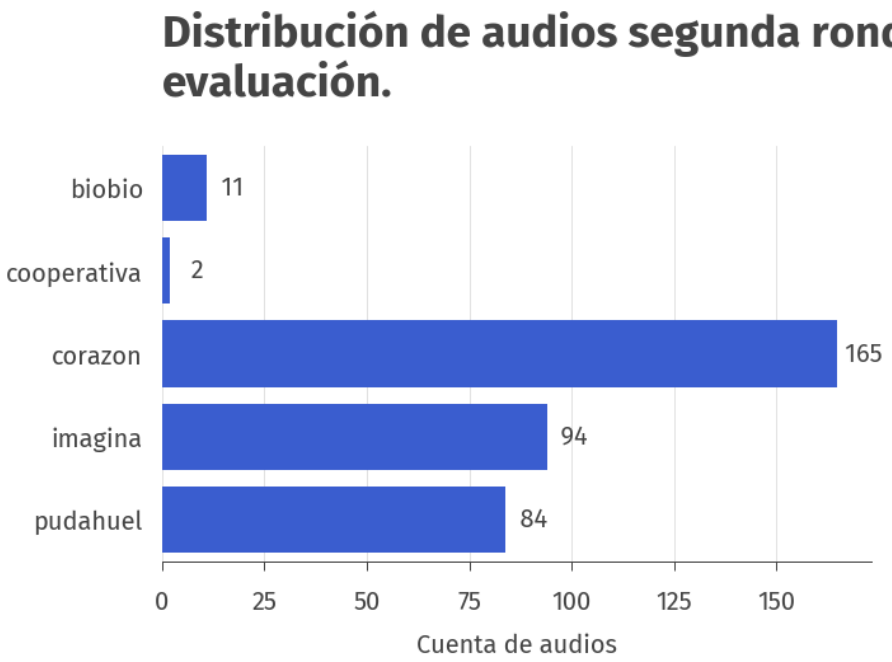


Figura 5.7: Distribución de radios de origen de las muestras en segunda ronda de evaluación.

5.3. Análisis de resultados

5.3.1. Enfoque 1

Respecto de los resultados obtenidos con el Enfoque 1 (sección 4.1.1), se deben destacar dos cosas: a pesar de que la tasa [9] obtenida era muy superior a la que se considera aceptada, es más económico mantener un programa (aunque lento) realizando clasificaciones por sobre una persona. Esto tiene una implicancia en la puesta en producción: independiente del tiempo

[9] Definida en 3.1.

de ejecución, dado que esta tarea no se realiza en tiempo real, es posible paralelizar las predicciones sobre nuevas grabaciones de radio. Esto abre la puerta a la utilización de este modelo en un entorno de producción. De todas formas no se considera por lentitud y uso poco eficiente de recursos computacionales. El desempeño no es considerablemente mejor que los obtenidos con métodos más simples.

Los resultados obtenidos con este enfoque, utilizando `AdaBoundW` como función de optimización en el proceso de entrenamiento, muestran peores resultados que los obtenidos sin esta función de optimización. En un principio se sospechó de un error en la implementación de la red neuronal, pero se corroboró el correcto funcionamiento con otra función de optimización de la misma red.

5.3.2. Enfoque 2

El desarrollo de este enfoque permitió mejorar considerablemente los tiempos de procesamiento. De hecho, es más de 800 veces más rápido que el enfoque 1. Por otro lado, en la clasificación obtiene resultados levemente peores. Otra ventaja importante en la implementación del enfoque dos, basado en distintos modelos simples de `sk-learn`, es el corto tiempo de entrenamiento. Esto permite una iteración más rápida entre entrenamiento y evaluación. De hecho, para el entrenamiento del enfoque 1, sin contar con la generación de las imágenes sólo en entrenamiento se utilizaban 12 hrs de GPU servidas en `Google Colab`. En el enfoque 2, en el lapso de algunos minutos el proceso de entrenamiento y *test* estaba completado.

Esto motiva a seguir desarrollando modelos en esta línea innovando en la selección de características (Enfoque 3).

5.3.3. Enfoque 3

Este modelo obtenía resultados similares en ambientes de entrenamiento y validación, manteniendo desempeño similar en ambientes de producción. Es por esto que se eligió este modelo para el entorno real. La forma de extraer características tomando en cuenta tanto los coeficientes de MEL como la serie temporal del sonido (comportamiento captado por las redes neuronales recurrentes LSTM), permite generar modelos más robustos que se comportan de forma similar tanto en entrenamiento como en producción.

Adicionalmente, la librería utilizada para la extracción de características se ha utilizado en distintos proyectos de identificación de hablantes. Esto gracias al aprendizaje generado en este trabajo.

5.3.4. Puesta en producción

Para analizar la puesta en producción se debe recordar el objetivo del proyecto: encontrar más fácilmente publicidades políticas. Después de dos rondas de validación y búsqueda, se observó que existen distintas formas de reducir el "área de búsqueda" de publicidades

políticas: una es utilizando el modelo desarrollado. Otro método corresponde seleccionando los horarios en los que más se reproducen los comerciales en la radio. Este conocimiento se deduce luego de analizar cientos de horas de grabaciones de radio.

La principal ventaja de utilizar un modelo de aprendizaje de máquinas corresponde a la automatización del proceso. Esto permite mecanizar la búsqueda, extracción de la muestra política y procesamiento de los audios seleccionados, sin perder calidad de audio. Esto porque anteriormente, una vez que se encontraba un comercial político, la forma de extraerlo correspondía a grabar con un segundo dispositivo (teléfono u otro computador) la salida del sonido del computador que contiene la publicidad encontrada. Adicionalmente, permite que una persona no experta o con pocos conocimientos computacionales pueda, sin dificultades, aportar en la búsqueda.

Capítulo 6

Conclusión y trabajo futuro

6.1. Conclusión

En primer lugar, se logra formular el problema de forma clara, cumpliendo así el primer objetivo específico. Esto permite acotar el alcance del trabajo y enfocar mejor los esfuerzos en diseñar distintos enfoques que permitieron clasificar audios y luego poner el mejor modelo en producción.

Se concluye que se cumple el objetivo general de la implementación de un programa en *Python* que, en base a procesamiento de señales y modelos basados en *machine learning* detecte comerciales. También se mejora la situación actual para la búsqueda de publicidades políticas mediante la automatización del proceso. Esto se logra utilizando el modelo desarrollado para las predicciones de comerciales y distintas plataformas que se aprendieron en el transcurso de la memoria: (*label studio*) para el etiquetado y búsqueda de publicidades políticas. En particular, en ambientes de producción se obtiene una *accuracy* de 90 % en búsqueda de comerciales en la radio bíobio observados en la primera ronda de validación, con un tiempo de procesamiento de menos de un minuto dada la tasa de procesamiento $0,04$. Es decir, se procesaron $11.640[s]$ de grabaciones en tan solo $46[s]$.

Es importante destacar que para el cumplimiento del objetivo general, se creó un *dataset* que permitiese entrenar modelos de *machine learning* en base a datos etiquetados. El *dataset* generado constaba de $1220[min]$ de audio, etiquetados manualmente con la herramienta *label studio*, entre 5 clases: comercial, entrevista, música, otros, *skipped*. El origen de las grabaciones correspondía a distintas radios, en distintos horarios permitiendo variedad en las muestras.

Respecto de la experiencia etiquetando datos, es importante destacar la importancia de utilizar una herramienta amigable. Para esto, se invirtió una considerable cantidad de tiempo (cercano a una semana) buscando una interfaz cómoda para el etiquetado. Esto es un aporte considerable para la empresa ya que permite reutilizar esta herramienta y los conocimientos generados en otras situaciones y problemas (que ocurre y se necesita constantemente).

Se concluye que los modelos basados en redes neuronales con espectrogramas como descriptores de las grabaciones son los que obtienen mejores resultados en ambientes de entre-

namiento. El desempeño de esta solución no varía considerablemente dependiendo del origen de la grabación pero los tiempos de procesamiento la alejaban de los objetivos generales y del objetivo específico que el programa debía ser más rápido que la solución actual.

Aún dada la lentitud (*tasa* 32.2) del modelo con redes neuronales profundas (enfoque 1), se debe destacar que la eficiencia de un modelo no es solo medible con la cantidad de aciertos y el tiempo de procesamiento. Para este trabajo, se creía inicialmente que era indispensable que el modelo superase a las personas en etiquetar datos. En realidad, solo se necesita que el sistema sea automático de forma que no haya dependencias de personas en el proceso. Solo esto, independiente de la velocidad del proceso, es una mejora de la situación actual en cuanto a la automatización de la búsqueda de publicidades en la radio.

Independiente de lo anterior, la tasa es una métrica importante en el desarrollo de la memoria a que adicionalmente al tiempo de ejecución, el programa era muy intensivo en el uso de memoria tanto RAM como disco duro, lo cual hace más caro el uso de recursos computacionales. Esto motivó a tomar decisiones como buscar modelos más eficientes en el transcurso de la memoria.

Los modelos implementados con `sk-learn` demostraron ser una gran herramienta para la rápida iteración y desarrollo. Las características extraídas mediante MFCC (*Mel Frequency Cepstral Coefficients*) con distinto ancho de ventana, permitían obtener descriptores de alto nivel con bajo tiempo de procesamiento, haciendo que los modelos como *Random Forest* o *SVM*, implementados con la librería mencionada, fueran la primera opción para la puesta en producción.

Ese enfoque tenía una desventaja que no se observó hasta el entorno de producción: los resultados de la segunda ronda de evaluación revelaron que había malos comportamientos para distintos orígenes de grabaciones. En particular, en comparación con la primera ronda de evaluación (que solo se compone de radio Bíobio) hay casi 40 puntos porcentuales de desempeño más que en la evaluación correspondiente a la segunda ronda, con variedad de radios de origen de grabaciones.

6.2. Trabajos Futuros

A continuación, se proponen algunos temas como trabajos futuros. Estos son complementarios a la presente memoria y pueden ayudar a mejorar distintos aspectos: desde la extracción de características hasta el estudio de los contenidos presentes en los comerciales.

- Extracción de contenido de los comerciales: mediante *software* de *speech to text* es posible obtener que se está diciendo en los comerciales. Esto permitiría clasificar si lo que se escucha es un comercial político, alimenticio o incluso encontrar marcas en particular.
- A nivel de procesamiento de señales: se propone el desarrollo de un descriptor de audio más robusto que los utilizados en esta memoria. Se propone verificar métodos similares a *Resemblyzer* o COLA [10].

[10] Contrastive learning of general purpose audio representations, basado en (Saeed, Grangier, y Zeghidour,

Se propone la creación de un dataset con metodologías más completas de muestreo. Se cree que el desempeño variable dependiente de la radio obtenido fue causado, en parte, por el desbalance en el origen de las grabaciones.

2020).

Bibliografía

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* (Software available from tensorflow.org)

[2] Collobert, R., Puhrsch, C., y Synnaeve, G. (2016). *Wav2letter: an end-to-end convnet-based speech recognition system.* arXiv 1609.03193.

[3] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009a). Imagenet: A large-scale hierarchical image database. En *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009b). Imagenet: A large-scale hierarchical image database. En *2009 ieee conference on computer vision and pattern recognition* (p. 248-255). doi: 10.1109/CVPR.2009.5206848

[5] Guzhov, A., Raue, F., Hees, J., y Dengel, A. (2020). *Esresnet: Environmental sound classification based on visual domain models.* arXiv 2004.07301.

[6] He, K., Zhang, X., Ren, S., y Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*.

[7] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., ... Wilson, K. (2017). *Cnn architectures for large-scale audio classification.* arXiv 1609.09430.

[8] Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. En F. Pereira, C. J. C. Burges, L. Bottou, y K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc.

[9] Loshchilov, I., y Hutter, F. (2017). *Decoupled weight decay regularization.* arXiv. doi: 10.48550/ARXIV.1711.05101

[10] Luo, L., Xiong, Y., Liu, Y., y Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. *CoRR*, *abs/1902.09843*.

[11] McFee, B., Lostanlen, V., Metsai, A., McVicar, M., Balke, S., Thomé, C., ... Kim, T. (2020, julio). *librosa/librosa: 0.8.0.* Zenodo. doi: 10.5281/zenodo.3955228

[12] Mollas, I., Tsoumakas, G., y Bassiliades, N. (2019, 11). *Lionforests: Local interpretation of random forests through path selection.*

[13] Palanisamy, K., Singhania, D., y Yao, A. (2020). *Rethinking cnn models for audio*

classification. arXiv 2007.11154.

[14] Papers With Code, b. M. A. R. (2022). *Papers with code: Image classification on imagenet.* https://paperswithcode.com/sota/image-classification-on-imagenet. (Accessed: 2022-04-3)

[15] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. En H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, y R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

[16] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

[17] Ronneberger, O., Fischer, P., y Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, *abs/1505.04597*.

[18] Saeed, A., Grangier, D., y Zeghidour, N. (2020). *Contrastive learning of general-purpose audio representations.*

[19] Simonyan, K., y Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition.*

[20] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2014). *Going deeper with convolutions.*

[21] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., y Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, *abs/1512.00567*.

[22] Tkachenko, M., Malyuk, M., Shevchenko, N., Holmanyuk, A., y Liubimov, N. (2020-2021). *Label Studio: Data labeling software.* (Open source software available from https://github.com/heartexlabs/label-studio)

[23] Unholster. (2020). *Unholster: Análisis de propaganda del plebiscito en las radios.* https://www.unholster.com/prensa/2020/11/20/cuando-y-dnde-los-patrones-diferentes-del-apruebo-y-rechazo-en-la-propaganda-del-plebiscito-en-las-radios. (Accessed: 2021-06-30)

[24] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi: 10.1038/s41592-019-0686-2

[25] Wan, L., Wang, Q., Papir, A., y Moreno, I. L. (2020). *Generalized end-to-end loss for speaker verification.*

[26] Yang, Y.-Y., Hira, M., Ni, Z., Chourdia, A., Astafurov, A., Chen, C., ... Shi, Y. (2021). Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*.