UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema De Telemetría Para Identificación De Patrones De Flujo En Tuberías Por Medio De Redes Neuronales

FI-DIE

Sello digital por emergencia sanitaria

Lic. Angélica Gutiérrez Vázquez
Coordinación de Titulación y Servicio Social
angelica_6@comunidad.unam.mx
20/09/2021

TESIS

Que para obtener el título de

Ingeniera en Computación

P R E S E N T A

María Fernanda Rocha Mancera

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Salvador Arce Benítez

DIRECTORA DE TESIS

Dra. Flor Lizeth Torres Ortiz

Ciudad Universitaria, Cd. Mx., 2021

Dedicatoria

Dedicada a ...

...a mis padres, Vale y Carlitos

por ser mis amigos, mis guías y mis héroes. Gracias por llevarme de la mano en este largo camino, nunca lo hubiera logrado sin su apoyo. Los amo,

...a mi hermano Milo

por estar para mi siempre que te necesito, darme tu apoyo incondicional y también por ser mi compañerito de conciertos,

... a ChocoYeti

por ser mi mejor amigo y acompañarme en esta travesía. Eres el mejor yeti. Sigamos siendo el mejor equipo que el mundo haya visto,

... a mi perrita Miranda

por enseñarme que el amor se puede demostrar sin necesidad de usar palabras

Con amor, Fernanda.

Dedicada a ...

...a mis padres, Adriana y Salvador

por haber estado atrás de mi, otorgándome todo el apoyo que siempre necesité, nunca dejando de creer en mi, y sobre todo por su amor incondicional,

...a mi hermana Sariruris

por ser mi amiga y por haber aguantado mi humor durante toda la carrera,

... a mi tía Rosita

por todo lo que me dio con el pasar de los años, por su amor incondicional, por los años de preparación y sobre todo por todo el cuidado que me dió,

... a Chocoprincesa

por ser la persona más especial en mi vida, por todas las aventuras y recuerdos que hicimos en toda la carrera, por haber aguantado la tesis conmigo aunque no quería hacerla y especialmente por todo su apoyo.

con amor, Salvador.

Agradecimientos

A nuestra tutora...

...la Dra. Lizeth Torres por todos los conocimientos que nos ha brindado, por su tiempo y dedicación los cuales son parte fundamental de nuestra formación profesional. Por ser una gran persona comprometida con su trabajo y motivarnos a concluir este trabajo. Por sus valiosos consejos y apoyo para seguir creciendo profesionalmente. Especialmente por darnos la oportunidad de escribir este trabajo.

...A la UNAM...

...por la oportunidad de estudiar en sus aulas, por todas las oportunidades que nos brindó y las maravillosas personas que conocimos en ella.

A la Facultad de Ingeniería...

...Por todos los conocimientos que nos brindó y que nos permitirán afrontar nuestra vida profesional, por ser nuestra segunda casa y por todos las momentos vividos y amistades forjadas dentro de sus paredes.

Al Instituto de Ingeniería...

...Por permitirnos aprender durante nuestra participación en este proyecto y así complementar nuestro desarrollo profesional.

Notación

- ACK - Acknowledged, Recibido
- ADC - Analog to Digital Converter, Convertidor Analógico Digital
- API - Application Programming Interface, Interface de Programación de Aplicaciones
- BN - Batch Normalization, Normalización del Lote
- BP - Back Propagation, Propagación Hacia Atrás
- CNN - Convolutional Neural Network, Red Neuronal Convolucional
- DC - Direct Current, Corriente Directa
- DCNN - Densely Connected Neural Network, Red Neuronal Densamente Conectada
- GPIO - Gerneral Purpose In/Out pins, Pins de Entrada/Salida de Propósitos Generales
- I^2C - Inter-Integrated Circuit, Circuito Inter-Integrado
- NACK - Not Acknowledged, No Recibido
- ReLU - Rectified Linear Unit, Unidad Linear Rectificada
- SCL - Serial Clock Line, Línea Serial de Reloj
- SDA - Serial Data Line, Línea Serial de Datos
- SGD - Stochastic Gradient Descent, Descenso de Gradiente Estocástico
- STFT - Short-time Fourier transform, Transformada de Fourier de Tiempo Corto

Índice general

Índice de figuras

Prólogo

Una de las ramas más complicadas de la mecánica de fluidos es el análisis y modelado de flujo bifásico, por lo que año con año se buscan nuevas tecnologías y herramientas que ayuden a incrementar y generar conocimiento en esta rama. En tal sentido, este trabajo de tesis presenta una nueva herramienta para aplicaciones que involucran flujo bifásico aire-glicerina (o flujos con propiedades similares). Esta herramienta consiste de dos partes: la parte principal es una red neuronal convolucional, cuyas entradas son espectrogramas generados a partir de diferenciales de presión. Esta red puede predecir los gastos másicos de la mezcla glicerina-aire que fluye por una tubería mediante un entrenamiento previo.

La segunda parte es la propuesta de un sistema de telemetría capaz de tomar lecturas de presión en un sistema de tuberías que transporta flujo bifásico, así como de guardar y enviar dichas lecturas a una plataforma web, de donde posteriormente podrán ser descargadas y procesadas para obtener imágenes de espectrogramas que sirvan de información de entrada para la red neuronal.

La herramienta que se propone en esta tesis puede ser la base para diseñar sistemas de diagnóstico, monitoreo o control para tuberías de flujo bifásico, ya que la herramienta propuesta puede funcionar como medidor de gastos másicos y ser utilizada para sustituir medidores que existen en el mercado actualmente.

Resumen

Esta tesis multidisciplinaria contiene seis capítulos. El primer capítulo presenta la introducción, que contiene el objetivo, la definición del problema, los antecedentes y la metodología.

En el segundo capítulo se explica detalladamente el funcionamiento de las redes neuronales convolucionales como procesos matemáticos y computacionales. Además, se describe cada una de las etapas que conforman estas redes.

En el tercer capítulo se discute el diseño e implementación de una red neuronal convolucional para la estimación de los gastos másicos de un flujo bifásico conformado por aire y glicerina. El entrenamiento de la red se basa en imágenes de espectrogramas generados a partir de mediciones de presión adquiridas en una tubería especializada, la cual se encuentra en el Instituto de Ingeniería de la UNAM (Hernández et al., 2019).

El cuarto capítulo presenta los resultados de la estimación ejecutada por la red neuronal después de varias iteraciones de sintonización de los parámetros de diseño. Los resultados, mostrados de manera gráfica, indican una mejoría entre cada iteración. También se describe la propuesta de un método para predecir la cantidad de imágenes necesarias para mejorar el desempeño de la red neuronal convolucional propuesta en esta tesis.

El quinto capítulo presenta la propuesta de una red de telemetría que es capaz de adquirir mediciones de presión mediante una computadora `Raspberry Pi 3` y enviarlas a una plataforma web para su posterior tratamiento. Y, finalmente, el sexto capítulo presenta algunas conclusiones sobre este trabajo de tesis.

Capítulo 1

Introducción

1.1. Objetivo General

Diseñar y entrenar una red neuronal convolucional para predecir los flujos másicos de un flujo bifásico glicerina-aire a presión en tuberías con imágenes de espectrogramas generados a partir de diferenciales de presión.

1.2. Objetivos Específicos

- Proponer una estrategia para optimizar el entrenamiento de una red convolucional con un número reducido de imágenes.
- Proponer una métrica, basada en regresiones lineales, para calcular el número de imágenes necesarias para entrenar una red neuronal convolucional.
- Proponer la arquitectura de un sistema de telemetría para el envío de mediciones de presión a una plataforma web tal que puedan ser descargadas y posteriormente procesadas y utilizadas como entradas de la red neuronal convolucional.

1.3. Definición del Problema

Cualquier combinación (sólido-líquido, sólido-gas o líquido-gas) en movimiento constituye flujo bifásico. Desde hace años se ha invertido un esfuerzo considerable en el flujo líquido-gas, ya que este tipo de mezcla es vital para el análisis y diseño de ciclos de refrigeración, centrales eléctricas, compresores, condensadores, evaporadores, producción de petróleo, calderas, e incluso, para la investigación de reactores nucleares y el desarrollo de tecnología espacial.

En la presente tesis se trata el flujo de sustancias en dos fases en tuberías horizontales, líquido-aire, ya que es una aplicación muy común en la industria petrolera, donde usualmente se transportan mezclas de hidrocarburos con gas.

El flujo bifásico en tuberías horizontales se comporta de diferente manera según el tamaño de la tubería, la velocidad del flujo, las propiedades del fluido y la inclinación de la misma. Debido a esta situación, para el diseño, análisis y desarrollo de sistemas de transporte, es fundamental determinar los diferentes comportamientos del flujo, llamados patrones de flujo, los cuales se pueden clasificar por sistema, apariencia y tipo (Caughron, 1967).

Clasificación por sistema: La clasificación por sistema categoriza a los flujos bifásicos en flujos con un solo componente (un líquido puro y su vapor) y en flujos con dos componentes con cualquier componente presente en una de sus fases.

Clasificación estructural: Un patrón de flujo se refiere a la distribución de fracciones volumétricas de las fases en una condición de flujo particular. Alves (1954) ilustró siete patrones de flujo para flujo horizontal: flujo con burbujas, flujo con tapones, flujo estratificado, flujo ondulado, flujo de ariete o pulsante, flujo anular y flujo disperso.

En vista de la complejidad que el flujo bifásico confiere a los procesos que lo involucran es muy importante desarrollar nuevas técnicas y tecnologías que ayuden al diseño óptimo de dichos procesos, a la seguridad de operación de las tuberías y a la resolución de problemas típicos que ocasiona el flujo bifásico. Es muy importante, entre otros aspectos, tener

información sobre las características del flujo bifásico como lo son: patrón de flujo, fase en equilibrio, velocidad de la fase, flujo crítico y factor de fricción.

Algunos de los problemas típicos que ocasiona el transporte de fluidos multi-fase son caídas de presión mayores que en flujo monofásico, patrones de flujo indeseables en ciertas condiciones (con problemas mecánicos de tubería, hidráulicos operacionales, etc.), erosión y corrosión por velocidades elevadas y poca precisión en cálculos hidráulicos (fricciones) con modelos aproximados. Para evitar estos problemas es fundamental disponer de información en tiempo real del flujo a dos fases, como por ejemplo el gasto másico del líquido y el gasto másico del gas. El problema es que la instrumentación para medir variables de flujo bifásico es sumamente costosa. En particular, los medidores de gasto para flujo bifásico son especialmente costosos, lo que puede elevar el precio de operatividad de las tuberías de transporte. Por esta razón, nuevas alternativas para medir los gastos de una manera económica siempre son deseables.

Una de estas alternativas, que es la que se propone en esta tesis, es el uso de transductores de presión, que son muy económicos y fáciles de instalar, además de que ya se han utilizado para identificar patrones de flujo bifásico (Matsui, 1984).

Con el fin de utilizar las mediciones de presión para estimar los gastos másicos de un flujo bifásico es necesario procesarlas, es decir, introducirlas como datos de entrada por un sistema (modelo) estocástico o determinista para obtener las salidas deseadas: los gastos másicos. En esta tesis se propone que este modelo sea una red neuronal convolucional y, puesto que este tipo de redes ha sido ampliamente investigado y mejorado recientemente, existe mucha información y muchos paquetes computacionales con librerías que permiten que estas redes sean implementadas fácilmente.

1.4. Antecedentes

La elección de una técnica de estimación de variables depende de muchos aspectos, como la simplicidad de la técnica, el costo de la instrumentación, del sistema de adquisición de

datos y del software para procesar los datos. Otros aspectos a tomar en cuenta son la resolución y precisión, así como la finalidad de la estimación. No es lo mismo estimar (identificar) para fines de diseño que para la detección de anomalías.

El uso de una Red Neuronal Convolucional (CNN de ahora en adelante por su nombre en inglés: *"Convolutional Neural Network"*) como herramienta de estudio y estimación de variables en aplicaciones que involucran la conducción de flujo bifásico en tuberías ya ha sido explorado. Por ejemplo, Du et al. (2018) proposieron una CNN para identificar los patrones de flujo de una mezcla petróleo-agua transportada en una tubería vertical. Para el entrenamiento de la CNN, se utilizaron imágenes adquiridas por una cámara de alta velocidad colocada frente a la tubería transparente. Para mejorar la adquisición de las imágenes se utilizó retroiluminación LED.

Un trabajo muy similar al que se propone en esta tesis fue presentado por Xu et al. (2020), quienes diseñaron una CNN para predecir los caudales, los patrones de flujo y la fracción vacía de gas de un flujo bifásico petróleo-gas. Para entrenar la red, los autores recopilaron tomografías por capacitancia eléctrica del flujo bifásico con diferentes caudales de petróleo y gas. Los instrumentos utilizados para adquirir las tomografías fueron colocados antes y después en un tubo venturi. A partir de los datos proporcionados por estos instrumentos, se obtuvieron imágenes utilizando un algoritmo basado en patrones binarios locales (LBP por el acrónimo en inglés de *"Local Binary Patterns"*). Según los autores, los resultados de la predicción utilizando imágenes del patrón de flujo como entrada de la CNN fueron mucho mejor que los resultados de la predicción utilizando como entrada los datos de capacitancia originales.

Torisaki and Miwa (2020) propusieron una CNN para extraer en tiempo real características de flujo multifásico burbujeante. A partir de la información de las coordenadas de cada burbuja detectada individualmente, se pueden adquirir instantáneamente características como la fracción vacía, la densidad del número de burbujas y el tamaño promedio de las burbujas.

Como puede verse, los trabajos mencionados son relativamente recientes. Esto signi-

fica que el uso de las CNN para el estudio y medición de parámetros que caracterizan el flujo bifásico es una línea de investigación en marcha, que puede aportar aún mucho conocimiento al área de mecánica de fluidos.

1.5. Metodología

Una CNN se denomina así por el concepto matemático convolución, que es una transformación lineal de dos funciones en una tercera. Debido a que las operaciones de convolución se realizan entre matrices bidimensionales, estas redes son muy efectivas para tareas de visión artificial, como la clasificación de imágenes, entre otras aplicaciones.

Los fundamentos de las CNN se basan en el artículo de Fukushima and Miyake (1982), el cual fue más tarde mejorado por LeCun et al. (1998) al introducir un método de aprendizaje basado en retropropagación para poder entrenar el sistema correctamente.

Dado que las entradas de las CNN suelen ser imágenes, y dado que en esta tesis se propone como método económico usar mediciones de presión para estimar los gastos másicos, las mediciones deben ser procesadas para ser transformadas en imágenes. Con este fin, las mediciones de presión se transformaron en diferenciales de presión. Después, utilizando la transformada de tiempo corto de Fourier (STFT por el acrónimo de *"Short-Time Fourier Transform"*), se construyeron espectrogramas a partir de los diferenciales, que son imágenes que muestran la evolución temporal del espectro de una señal. Finalmente, los espectrogramas fueron convertidos en archivos con formato de imagen, que se guardaron en carpetas debidamente etiquetadas, para después servir como entradas de la CNN. Un esquema de esta metodología se presenta en la Figura 1.1.

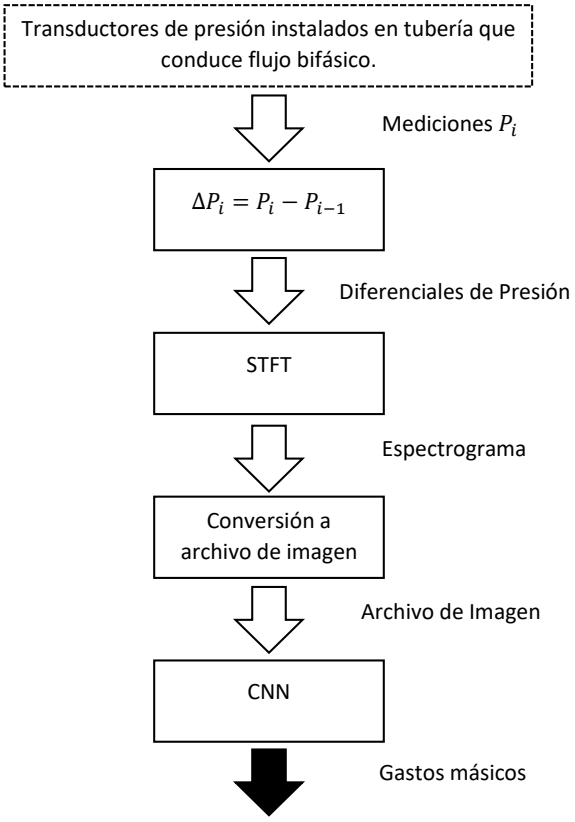


Figura 1.1: Método para estimar los gastos másicos en una tubería que conduce flujo bifásico glicerina-aire.

Capítulo 2

Redes Neuronales Convolucionales

2.1. Introducción a las Redes Neuronales

La redes neuronales artificiales son modelos computacionales que se utilizan para predecir. Dicho de manera más concreta, son un conjunto de algoritmos matemáticos y computacionales, cuyo diseño está inspirado en las neuronas biológicas del cerebro humano, que traducen información de entrada en una predicción. Una red neuronal tiene la finalidad de reconocer patrones mediante operaciones matemáticas y mediante la interpretación de datos (numéricos, de audio, de imágenes) a través del etiquetado y agrupamiento de datos en bruto. Los patrones que se reconocen son numéricos y están contenidos en una serie de vectores, dentro de los cuales es posible introducir todo tipo de datos y mediciones del mundo real, sean estos datos números, imágenes, sonido o texto.

Las redes neuronales ayudan a agrupar información sin etiquetar de acuerdo a características similares. Para poder realizar esta clasificación necesitan un conjunto de datos previamente etiquetados sobre los cuales entrenar.

Una red neuronal está conformada por la unión de neuronas de entrada (datos de entrada) con neuronas de salida (las predicciones finales) a través de capas ocultas y operaciones matemáticas.

Por ejemplo, una sencilla red neuronal es la que se conforma de dos neuronas de entrada conectadas con una sola de salida, mediante un conjunto de procesos matemáticos, como se muestra en la Figura 2.1. El proceso sería el siguiente: el dato de entrada x_1 se multiplica por un valor, denominado peso w_1, y llega a la capa siguiente. Por otro lado, el valor de entrada x_2 se multiplica por un peso w_2 y llega a la siguiente neurona donde se le suma el valor de la primer operación, y adicionalmente se suma un valor de sesgo o *"bias"* b_1. Finalmente, a este valor resultante se le aplica una función de activación (explicada más adelante) y este nuevo resultado es el valor de salida de esta capa. Dado que en este ejemplo la red es solo de dos capas, este valor final es la predicción. La operación matemática está expresada por la siguiente ecuación:

$$y_1 = f(x_1 w_1 + x_2 w_2 + b_1), \tag{2.1}$$

donde y_1 es la predicción.

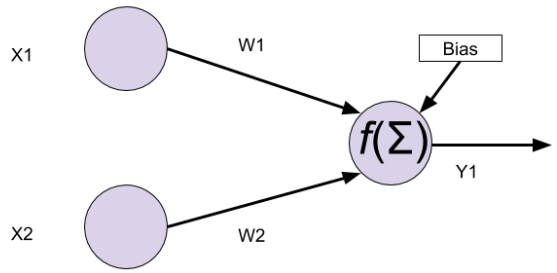


Figura 2.1: Ejemplo simple de una red neuronal.

La red neuronal anterior es realmente intrascendente, porque una verdadera red neuronal posee muchas más entradas y varias capas escondidas antes de llegar a la predicción del valor final y, especialmente, porque aún falta explicar como guardan y corrigen el conocimiento las redes neuronales, cosa que se explicará más adelante.

2.2. Redes Neuronales Convolucionales

Existen múltiples tipos de redes neuronales y se clasifican bajo diversos criterios. Para este trabajo es de particular interés el caso de las llamadas Redes Neuronales Convolucionales.

La función principal de una CNN es la de, mediante un entrenamiento previo, poder clasificar principalmente imágenes, de tal forma que cuando se le presente una imagen no utilizada en su entrenamiento sea capaz de predecir a qué conjunto de imágenes pertenece. Por ejemplo, una CNN con un entrenamiento de imágenes de perros y gatos podrá clasificar la imagen de un gato (correcta o incorrectamente, dependiendo de su entrenamiento), y si se presenta una imagen que no sea de ninguno de los conjuntos intentará dar una clasificación. En la realidad, una CNN tiene un alcance mucho mayor que solo la clasificación de imágenes. Si los datos que se le presentan para clasificar están en un arreglo útil, las CNN pueden clasificar todo tipo de datos. Sin embargo, el trabajo con imágenes es el más común.

De manera general podemos decir que una red neuronal consta de tres tipos de capas: capas convolucionales (las cuales dan el nombre a la red); capas de *"pooling"*, las cuales tienen la finalidad de reducir la cantidad de parámetros al quedarse con las características más comunes; y capas densamente conectadas, las cuales son las encargadas de producir el resultado final de la red. Cada una de estas capas se verá con mayor detalle en las subsecciones siguientes. Sin embargo, es importante mencionar que la estructura de la red consta de capas convolucionales seguidas de una capa de *"pooling"*. La cantidad de capas convolucionales pueden variar (generalmente una o dos), pero siempre será una sola de *"pooling"* tras las anteriores. Las capas densamente conectadas siempre serán la etapa final de la red pues son la parte que realiza la clasificación.

2.2.1. Funcionamiento de las Redes Neuronales Convolucionales

Las CNN son un tipo de red neuronal cuyo nombre se deriva del uso de la operación lineal conocida como convolución.

Una característica sumamente importante de este tipo de redes es que funciona principalmente para la identificación y la clasificación de imágenes. Nuevas investigaciones han acoplado las CNN para reconocimiento de otros tipos de datos, como el texto, sin embargo estas nuevas implementaciones van más allá del alcance de esta tesis. Cuando trabajan en su campo más común, las entradas son imágenes, ya sea en escala de grises o a color.

Al igual que otras redes neuronales, las CNN están conformadas por etapas. Para el caso de las CNN que reconocen imágenes es importante recordar que una imagen digital es una matriz de píxeles conformada por colores del modelo RGB. El modelo RGB es un modelo de color aditivo que codifica colores como combinaciones de tres colores primarios: rojo, verde y azul (R,G y B). Que el modelo sea aditivo significa que todo color empieza en negro y es creado al sumar colores primarios. Para crear distintos colores, se debe modificar la intensidad del brillo de cada color primario de manera independiente. Estas intensidades controlan el tono y la brillantez del color resultante. Para la mayoría de imágenes digitales, los valores RGB que conforman una imagen son positivos y se encuentran en el rango $[0, C_{max}]$, con $C_{max} = 255$, en otras palabra cada píxel de un canal RGB se codifica con 8 bits. Cada posible color C_i corresponde a un punto que se encuentra en la matriz cúbica o cubo de color RGB, de la forma $C_i = (R_i, G_i, B_i)$, donde $0 \leq \{Ri, Gi, Bi\} \leq C_{max}$ (Burger and Burge, 2016).

Cada píxel de una imagen en cualquier punto de la CNN deja de denominarse como píxel y se conoce como parámetro o característica. Posteriormente esta primer imagen de entrada se convoluciona con un *"kernel"* o filtro, es decir, se filtra la imagen.

A continuación se describe a detalle cada etapa que conforma el proceso de una CNN.

2.2.2. Convolución

En matemáticas, una convolución es un operador matemático que opera dos funciones f y h para transformarlas en una tercera y. Dicho operador se representa con el asterisco $(*)$ y la operación entre las dos funciones se expresa de la siguiente forma:

$$y(t) = (f * h)(t) = \int_{-\infty}^{\infty} f(\lambda)h(t-\lambda)d\lambda \tag{2.2}$$

La convolución es un operador que tiene aplicaciones en múltiples áreas de la ciencia y la ingeniería de donde destacan: procesamiento digital de imágenes, procesamiento digital de datos, química analítica, ingeniería eléctrica, física, teoría de probabilidad, radioterapia, entre otras (Behl et al., 2014).

Las CNN se llaman así, por que el primer paso de su algoritmo es una suerte de convolución de dos matrices. Por tanto esta convolución es, en primer lugar, discreta (no expresada por una integral, sino como una suma) y en segundo lugar, por ser matricial es en dos dimensiones. De manera matemática, la ecuación 2.3 (Goodfellow et al., 2016) representa dicha convolución matricial:

$$B_k(i,j) = (F_k * A)(i,j) = \sum_{l=0}^{nH}\sum_{m=0}^{nW} F_k(l,m)A(i-l, j-m), \tag{2.3}$$

donde $*$ denota convolución, B_k es la imagen filtrada o convolucionada, F_k es la imagen que se convoluciona, A es el núcleo de convolución (*"kernel"* o matriz que convoluciona y filtra), i es la dimensión renglón de las matrices, j es la dimensión columna de las matrices, nH es el alto de la imagen a filtrar, nW es el ancho de la imagen a filtrar. Finalmente, m y l son las variables contadoras (índices).

En la Figura 2.2 se puede apreciar la convolución entre una imagen de 3×3 píxeles a color (3 canales) y 3 *"kernels"* para conformar una imagen filtrada, también denominada mapa de características (*"feature map"*) o mapa de activación (*"activation map"*) (Torres,

2018). Se denomina de estas dos formas por que una imagen que se haya filtrado es una imagen donde se han resaltado o activado características especiales de la misma, como lo son bordes, curvas, círculos, entre otros.

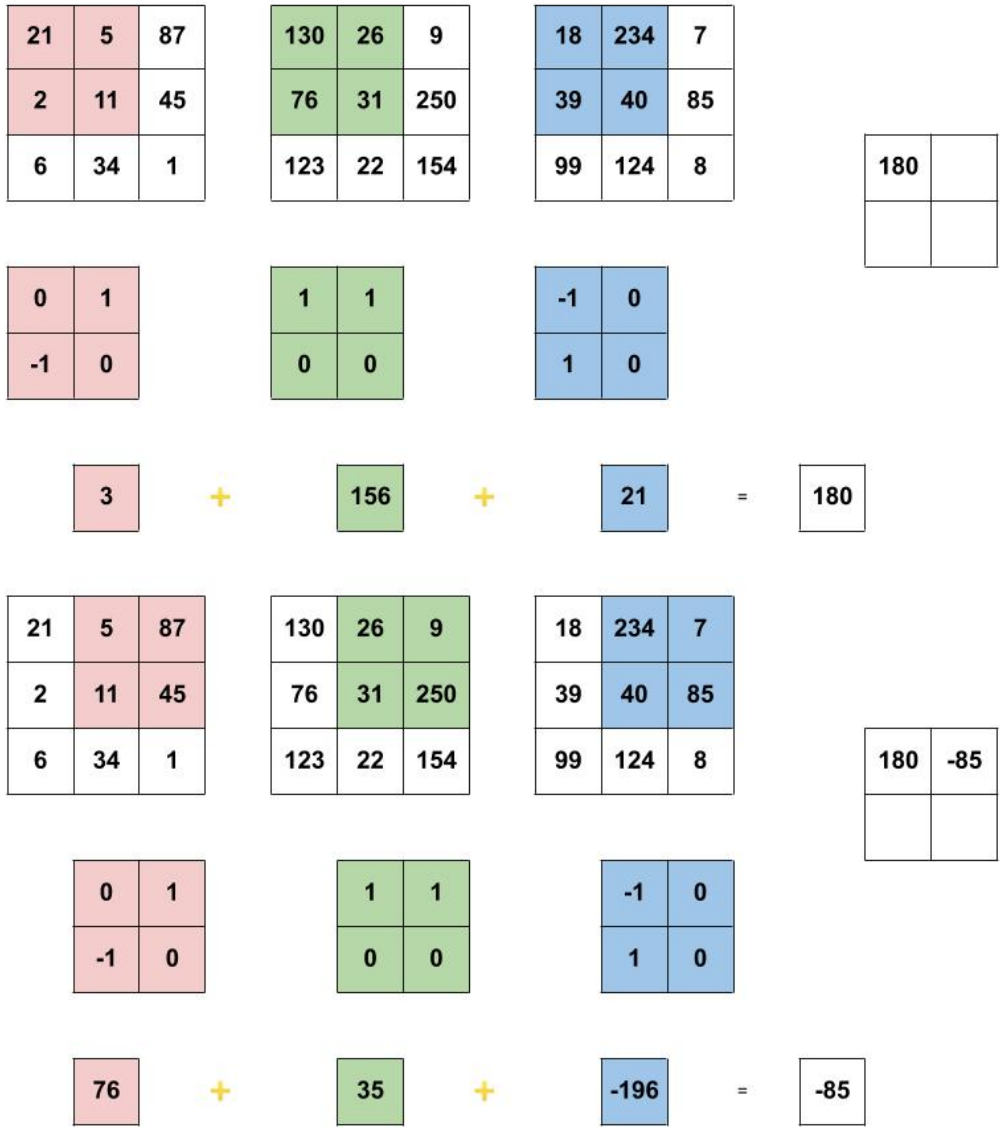


Figura 2.2: Ejemplo de convolución. A cada canal de una imagen RGB se le aplica una convolución con un *"kernel"* diferente. Esos 3 resultados se suman y se vuelven un valor de la imagen filtrada.

El objetivo de convolucionar con *"kernels"* es el de resaltar detalles o características de la imagen que se busca filtrar. Diversos valores de los números que conforman un *"kernel"* generan filtros que resaltan diversos detalles en una imagen, como lo son bordes, esquinas,

líneas curvas, círculos, líneas rectas, entre otros. En la Figura 2.3 se pueden apreciar algunos ejemplos de filtros y sus efectos en una imagen.

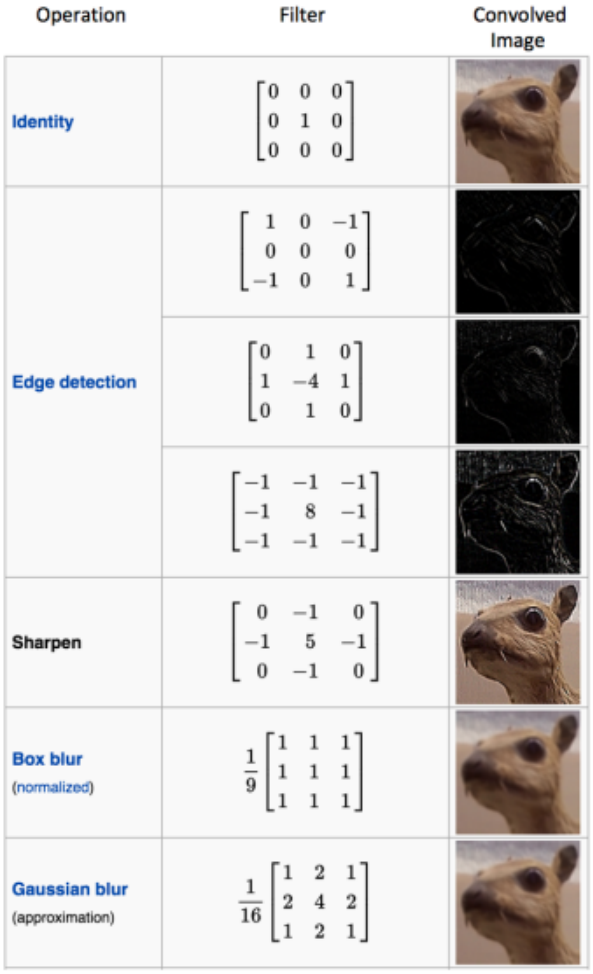


Figura 2.3: Ejemplos de filtros (Prabhu, 2018).

2.2.3. *"Kernel"*

Un *"kernel"* (o filtro) es una matriz de tamaño reducido pero configurable cuyos elementos son números a los que llamaremos pesos W. Estos números pueden ser pre-seleccionados o pueden ser números aleatorios. El uso de números aleatorios al inicializar los *"kernels"* es una práctica más común y ha demostrado un desempeño favorable (Goodfellow et al., 2016). La CNN de esta tesis usó inicialización aleatoria. En el ámbito más técnico, estos pesos son pseudo-aleatorios ya que se inicializan (en Python) con una función generadora

de distribución probabilística llamada `glorot_uniform`, la cual toma valores en un rango dado por una distribución uniforme, cuyos límites se expresan por la Ecuación 2.4 (Glorot and Bengio, 2010).

$$W \sim U\left(\sqrt{\frac{6}{n_{in}+n_{out}}}, -\sqrt{\frac{6}{n_{in}+n_{out}}}\right), \tag{2.4}$$

donde n_{in} es el tamaño de la imagen de entrada y n_{out} el tamaño de la imagen de salida.

El proceso de una CNN inicia con un conjunto de imágenes de entrenamiento. Este proceso comienza con una imagen inicial, la cual se convoluciona con un primer *"kernel"*. Una vez que se convoluciona esta imagen inicial con el *"kernel"* número uno, se tiene a la salida una imagen filtrada. Posteriormente, se filtra con un segundo *"kernel"*. Este proceso se repite con tantos *"kernels"* se desee.

Cada vez que una imagen se convoluciona con un *kernel*, o se filtra, disminuye su tamaño. Esto es debido a que cuando sucede la convolución, el kernel se opera con una cantidad de pesos iguales a los suyos, para resultar siempre en un solo peso. Por ejemplo, si un kernel es de 2×2, al convolucionar con una imagen operará sus 4 pesos con 4 parámetros de la imagen, resultando en un solo peso a la salida.

Un factor importante, es que la cantidad de información de los píxeles en el centro de una imagen es superior a los píxeles a la orilla, aún así, estos últimos poseen información. Para que la información de las orillas no se vea desperdiciada y pueda aprovecharse para el entrenamiento de la CNN, se usa un relleno o *"padding"*, el cual es un recubrimiento o borde extra a la imagen, de tal forma que cuando suceda el filtrado, la operación se realice con toda la imagen, y no se ignore el borde. En la práctica, este borde se conforma de ceros. En la Figura 2.4 se ilustra gráficamente en una 'imagen' de 4 píxeles de tamaño la razón de uso del *"padding"*.

Para calcular el tamaño o grueso del *"padding"*, se utiliza la Ecuación 2.5, donde f es

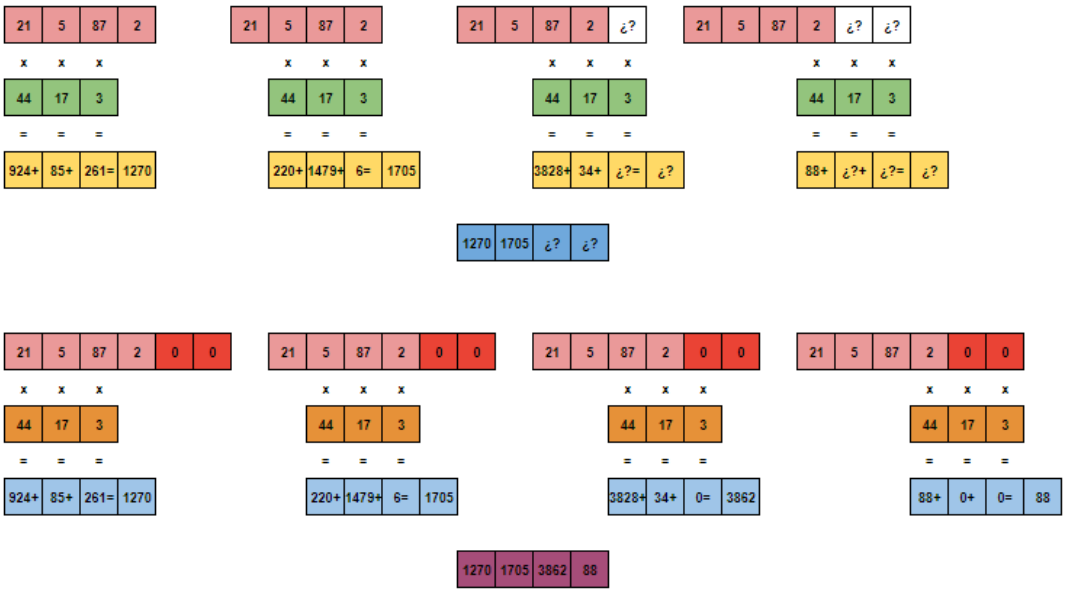


Figura 2.4: *"Padding"*. En esta imagen de 4 píxeles, que se convoluciona con un filtro de 3 píxeles, es necesario utilizar *"padding"*, de lo contrario se pierde información. Esto se debe a que la convolución debe efectuarse donde existen números.

la dimensión del *"kernel"* (Skalski, 2019).

$$p = \frac{f-1}{2} \tag{2.5}$$

La convolución puede suceder en pasos o *"strides"* de diferentes valores, es decir, cuántos valores hacia el lado y hacia abajo se mueve el *"kernel"* cuando convoluciona con la matriz de la imagen. Para calcular el tamaño de la imagen filtrada, a partir de tamaño del *"kernel"*, y del paso o *"stride"* se utiliza la Ecuación 2.6 (Skalski (2019)), donde n_{in} es el tamaño de la imagen de entrada, p es el *"padding"*, s es el *"stride"* y f es la dimensión del *"kernel"*.

$$n_{out} = \frac{n_{in} + 2p - f}{s} + 1 \tag{2.6}$$

Esto concluye la primer etapa de convolución, donde una imagen inicial se transforma en n imágenes filtradas, siendo n el número de *"kernels"* iniciales, regularmente potencias de 2. Dependiendo del tamaño de los *"kernels"*, será el tamaño de las imágenes de salida.

También depende de si la imagen es a color o en escala de grises.

2.2.4. Función de Activación

Una vez que la etapa anterior de la red es completada, el *"stack"* o volumen de imágenes debe pasar por una capa conocida como función de activación (*"activation function"*). La función de activación tiene como objetivo decidir si un parámetro continúa su camino por la red o no, como si fuese un *"switch"*. Esto con el objetivo de filtrar los parámetros que no poseen suficiente información, de aquellos que sí la tienen. Esto se realiza operando el parámetro en cuestión (recordando que un parámetro es un número) en una función específica. Existen muchas funciones de activación, cada una con su ecuación, rangos de valores de salida asociados, características y desempeños. Algunos ejemplos de funciones de activación se observan en la Figura 2.5. Aquí se pueden observar las reglas de correspondencia de dichas funciones y, gracias a las gráficas, podemos darnos una idea de los valores de salida, basados en los valores de entrada.

De todas estas funciones de activación son de especial interés dos de ellas: la función ReLU y la función softmax.

La función ReLU (del inglés *"Rectified Linear Unit"*: Unidad Lineal Rectificada) es una función de activación cuya regla de correspondencia se aprecia en la Ecuación 2.7. De todas las funciones de activación que se usan en las CNN, esta es la que tiene mejores resultados (Brownlee, 2019).

$$ReLU(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases} \tag{2.7}$$

La función softmax o función exponencial normalizada es una función de activación especial por que su salida es utilizada como una distribución de probabilidad sobre sus entradas, es decir, asignará una probabilidad de salida a cada valor de un vector de entrada. De esta, se hablará un poco en la Sección 2.2.9. Capa Completamente Conectada.

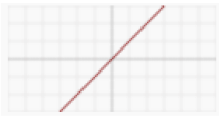
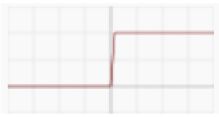
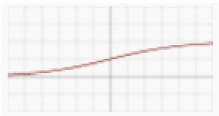
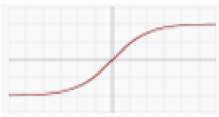
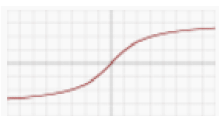
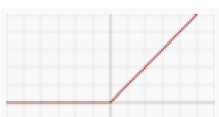
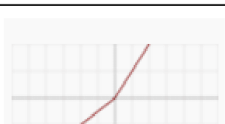
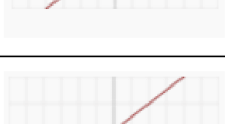
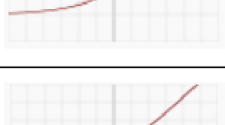
Nombre	Gráfica	Ecuación	Derivada
Identidad		$f(x) = x$	$f'(x) = 1$
Escalón Binario		$f(x) = \begin{cases} 0 \; para \; x < 0 \\ 1 \; para \; x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \; para \; x \neq 0 \\ Ind. \; para \; x = 0 \end{cases}$
Paso Suave (logarítmica)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
ReLU (Función unitaria linear rectificada)		$f(x) = \begin{cases} 0 \; para \; x < 0 \\ x \; para \; x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \; para \; x < 0 \\ 1 \; para \; x \geq 0 \end{cases}$
PReLU (Función unitaria linear rectificada paramétrica)		$f(x) = \begin{cases} \alpha x \; para < 0 \\ x \; para \; x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \; para \; x < 0 \\ 1 \; para \; x \geq 0 \end{cases}$
ELU (Función unitaria lineal exponential)		$f(x) = \begin{cases} \alpha(e^x - 1) \; para \; x < 0 \\ x \; para \; x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \; para \; x < 0 \\ 1 \; para \; x \geq 0 \end{cases}$
SoftPlus		$f(x) = ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figura 2.5: Diversas funciones de activación. Se ilustran las gráficas, la ecuación que les da forma y sus derivadas (Sharma, 2017).

2.2.5. *"Pooling"*

Dado que una CNN se entrena con una gran cantidad de imágenes, y que aumenta la cantidad de estas imágenes de acuerdo a la n cantidad de *"kernels"*, la información que maneja una CNN se incrementa en cada etapa. Es por ello que tras la convolución sigue una etapa llamada *"pooling"*.

El *"pooling"* (agrupación) toma una imagen y la segmenta. Lo hace agrupando píxeles (parámetros). Esta agrupación no es aleatoria pues depende del tamaño de la agrupación de los píxeles ($n \times n$ píxeles) y depende también de un *"stride"* (que es definido en esta etapa y no es el *"stride"* del convolucionado). Posteriormente, a estas matrices les sucede una de dos cosas, dependiendo del tipo de *"pooling"* asignado. Si el *"pooling"* elegido es *"max pooling"*, entonces de cada grupo de números, el número a la salida será el número más grande de la agrupación. Si el *"pooling"* escogido es *"average pooling"*, entonces se promediarán los valores dentro del grupo y este promedio será la salida. Un ejemplo de los tipos de *"pooling"* se ilustran en la Figura 2.6. Por otro lado, en la Figura 2.7 se ejemplifica el efecto que se obtiene al utilizar distintos valores de *"stride"*.

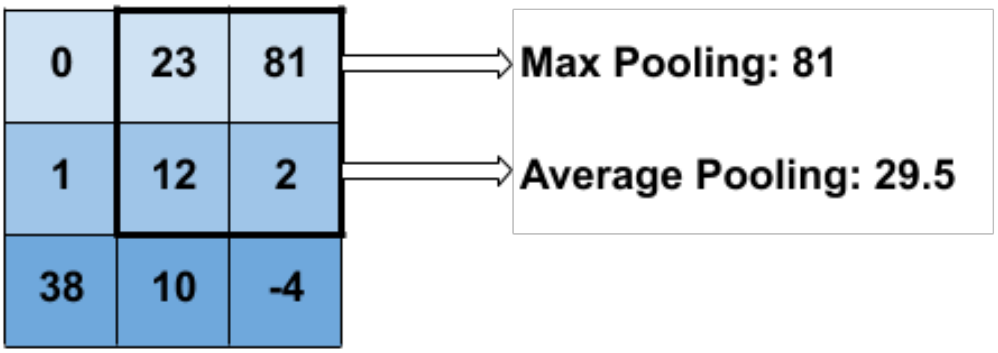


Figura 2.6: Ejemplo de los 2 tipos de *"pooling"*: *"Average"* y *"Max"*.

De esta forma, es posible abstraer la mayor información de cada pequeña parte de cada imagen, mientras la imagen reduce su tamaño.

La reducción del tamaño está en función de las configuraciones del *"pooling"*, pero de

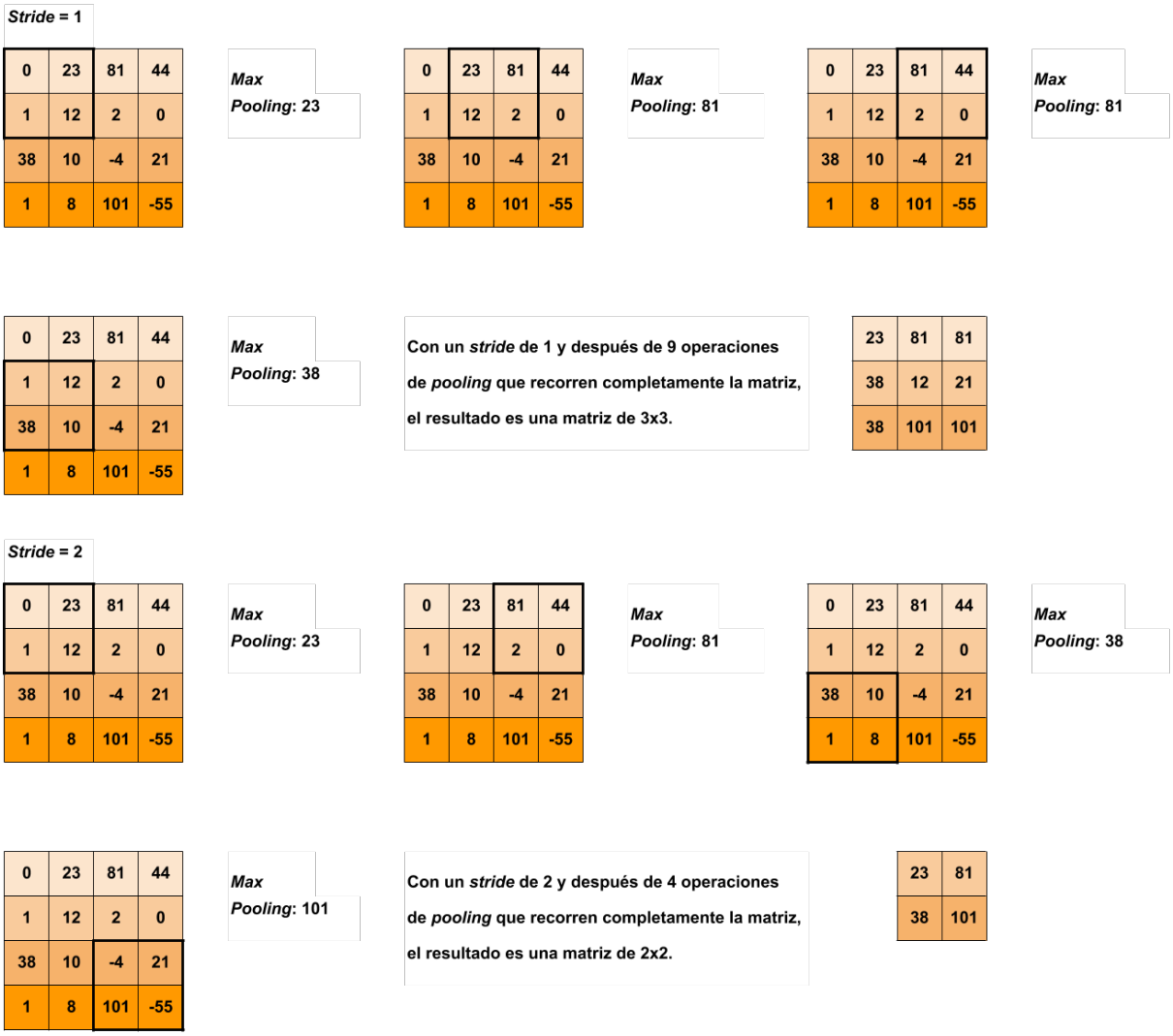


Figura 2.7: Diferencia entre utilizar *"stride"* con valor de 1 y con valor de 2.

manera regular se usan conjuntos de 2×2 píxeles de tamaño, con *"stride"* de 2 (es decir, un número ya agrupado no forma parte de otro grupo), de manera que las imágenes se reducen a la mitad de su tamaño. Por ejemplo, una imagen de 8×8 píxeles se volverá de 4×4. También por experiencia empírica en el ámbito de las CNN, es sabido que el uso de *"max pooling"* es más eficaz que *"average pooling"* (Torres, 2018). Esto se debe a que el efecto del promedio de *"average pooling"* suaviza la imagen. Esto quiere decir que características prominentes podrían ser suavizadas y por tanto no detectadas por la red (Basavarajaiah, 2019).

Esta etapa de una CNN puede entenderse como el resumen de una imagen filtrada o resumen del mapa de características (*"feature map"*).

2.2.6. *"Dropout"*

Tras varias fases de convolución y *"pooling"*, dependiendo del diseño de la CNN, la etapa posterior se conoce como *"dropout"*. Es un paso que se pone por recomendación o por regla de costumbre, pero puede quitarse o ponerse y observar los resultados.

La capa de *"dropout"* o abandono, es una capa que se encarga de apagar o retirar neuronas. En una CNN, con cada etapa del proceso, se van generando más y más parámetros, lo cual se vuelve un problema considerando que el equipo de cómputo donde sucede el proceso no tiene recursos infinitos. Por otro lado, usar *"dropout"* evita que en la red se produzca *"overfitting"*.

El *"overfitting"* (y su contraparte el *"underfitting"*) son errores que se pueden presentar a la hora de entrenar CNN. No son de errores de cálculo o programación, sino resultados que indican una mala predicción. Estos errores se describirán más adelante en este mismo capítulo y se observarán gráficamente en el capítulo 4 Resultados de la Red Neuronal.

La capa de *"dropout"* simplemente selecciona parámetros al azar y los 'apaga' o 'abandona' (*"drops them out"*). De esta forma, la carga de parámetros es menor y en total se tienen que consumir menos recursos en la computadora.

Como se mencionó antes, el uso de esta capa es empírica. Muchas CNN la utilizan y otras no. Para el caso particular de la CNN descrita en esta tesis, se probó utilizarla u omitirla. Los resultados de utilizar *"dropout"* fueron ligeramente mejores y se optó por dejar la capa.

2.2.7. *"Batch Normalization"*

La capa siguiente, al igual que la anterior, es una capa opcional. La capa de *"batch normalization"* no es una capa sencilla. Esta capa tiene como objetivo combatir un concepto conocido como cambio covariable (*"covariate shift"*). Cuando una red neuronal (convolucional o tradicional) se entrena, se hace con un conjunto de datos, sin embargo, muchas veces estos datos no vienen de la misma fuente, o tienen diferencias tales que es posible agrupar estos datos en subconjuntos. Por ejemplo, si se tiene una CNN que busca diferenciar bananas de entre otras frutas, el conjunto de imágenes de frutas puede contener imágenes de bananas maduras, no maduras, abiertas e inclusive bananas ya pasadas. Y a pesar de esto, la CNN debe identificarlas como bananas y discernirlas de otras frutas. El cambio covariable se presenta cuando los *"mini-batches"* de la CNN (es decir, los pequeños conjuntos de imágenes que entrena al mismo tiempo) están conformados por una más elevada cantidad de imágenes de cierto tipo, sobre de otro. Una vez más con el ejemplo, el cambio covariable se presenta si ciertos *"mini-batches"* tiene más imágenes de bananas maduras o muy maduras, en vez de tener una distribución aleatoria uniforme de los tipos de imágenes asociados con la etiqueta 'banana'. Lo que genera entonces el cambio covariable es un aprendizaje más lento (Nayak, 2018).

Estos desniveles entre los *"mini-batches"* se vuelven más difíciles de corregir en las capas ocultas de una CNN. Es por ello que se creó la normalización por lotes (en inglés *"batch normalization"*). Esta normalización tiene la función de mantener los pesos de la CNN en cada *"mini-batch"*, para evitar que haya mucho movimiento de parámetros. Modifica el *"bias"*, tal que cuando el optimizador elegido (del cual se hablará más adelante) comience a hacer su trabajo, se genere un ajuste de parámetros que evite que los pesos entre un *"batch"* que ya pasó y el *"batch"* siguiente no sean tan distintos. Los cálculos y fórmulas asociadas a esta capa pueden verse más a detalle en el trabajo de Ioffe and Szegedy (2015) donde se propone originalmente.

2.2.8. *"Flattening"*

El paso siguiente en una CNN es conocido como aplanamiento o *"flattening"*, el cual es el proceso de 'aplanar' el volumen de imágenes filtradas. Una imagen de entrada, conforme avanza en el proceso de convolución, de *"pooling"* y de *"dropout"* se transforma en cientos de imágenes, dependiendo de la cantidad de filtros y la cantidad de las capas de la CNN. Sin embargo, este volumen de imágenes filtradas o mapas de características no es una entrada adecuada para la etapa final de la CNN. Esta etapa final es una red neuronal densamente conectada (DCNN por su nombre en inglés: *"Densely Connected Neural Network"*). La tarea de la capa *"flattening"* es aplanar el volumen de imágenes en un vector unidimensional, el cual ya puede ser usado como entrada para la parte final de la red. Cada parámetro del volumen de imágenes o *"stack"* se vuelve un parámetro de este vector de valores de entrada. De esta manera una imagen puede entrar en la capa completamente conectada de la CNN. El proceso de *"flattening"* se ilustra en la Figura 2.8.

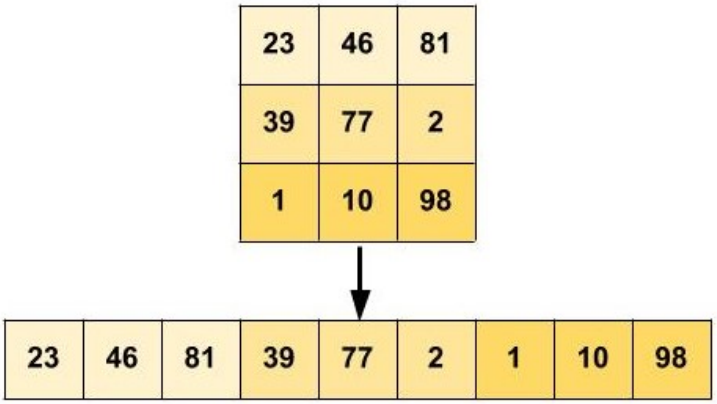


Figura 2.8: Proceso de *"flattening"* o aplanado.

2.2.9. Capa Completamente Conectada

En la etapa final de la CNN, los datos entran en una DCNN, donde los datos son usados para el aprendizaje mediante un proceso conocido como *"backpropagation"* o bien retropropagación.

El vector final 'aplanado' de imágenes se vuelve la capa de neuronas de entrada de una DCNN (que son las redes neuronales clásicas). Aquí cada elemento del vector aplanado se vuelve una neurona de la capa inicial y debe conectarse con cada neurona final de la capa de salida. Esta conexión sucede mediante operaciones con pesos y *"bias"*, como se explicó al comienzo de este capítulo. Lo normal es que existan capas intermedias, denominadas ocultas, entre las neuronas de entrada y las de salida. Esto se puede entender mejor con un ejemplo. Supongamos un vector 'aplanado' conformado por diez elementos $x = (x_1, x_2, ...x_{10})$, que es salida de las etapas anteriores y es entrada de la sección densamente conectada. Este ejemplo de red identifica entre tres clasificaciones, es decir, la capa de salida solo es de tres neuronas. También supongamos, por motivos de explicación, que no hay capas ocultas. Lo que debe pasar es cada elemento de nuestro vector de entrada debe conectarse con la capa de salida mediante la multiplicación de pesos y la suma de un solo *"bias"*. La neuronal de salida uno se vería de la siguiente forma:

$$y_1 = x_1w_1 + x_2w_2 + ... + x_{10}w_{10} + b_1 \tag{2.8}$$

Luego, el vector 'aplanado' debe conectarse con la segunda neurona de salida de manera semejante. Es importante mencionar que los pesos que conectan el vector aplanado con cada una de las neuronas de salida son todos diferentes y en la primer iteración de todo el proceso de la CNN son valores aleatorios. La conexión entre el vector 'aplanado' y la segunda neurona de salida sería la siguiente:

$$y_2 = x_1w_{11} + x_2w_{12} + ... + x_{10}w_{20} + b_2 \tag{2.9}$$

Finalmente, la conexión entre el vector 'aplanado' y la tercer neurona de salida sería la siguiente:

$$y_3 = x_1w_{21} + x_2w_{22} + ... + x_{10}w_{30} + b_3 \tag{2.10}$$

Cuando todas estas operaciones suceden, los resultados no se operan en una función de activación regular como se había hecho en etapas anteriores. Lo que se sucede es que

las neuronas resultantes ($y_1,\ y_2,\ y_3$) se vuelven los elementos y_i ($\forall i = 1, 2..., K$) de un vector $Y \in \mathbb{R}^K$, y este vector se opera mediante una última función de activación especial denominada Softmax. La expresión de la función Softmax se observa en la Ecuación 2.11 (Versloot, 2020).

$$\begin{aligned} \text{Softmax} : \mathbb{R}^K &\longrightarrow [1,0]^K \\ \text{Softmax}(y_i) &= \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}, \end{aligned} \tag{2.11}$$

donde y_i es el i-ésimo elemento del vector de neuronas de salida, $\sum_{j=1}^{K} e^{y_j}$ es la suma de la exponencial de los elementos del vector de entrada y K es el tamaño del vector de entrada.

Para cada valor del vector de entrada, la salida de la función softmax será el exponente de cada valor individual, dividido entre la suma de los exponentes de todas las entradas. Todo valor negativo se vuelve positivo. La división entre la suma genera un promedio, de tal forma que todo valor de salida está en el rango de 0 a 1. Los valores de salida son proporcionales entre ellos.

Esta función de activación es especial porque a cada resultado final le otorga un valor que puede asociarse con una distribución de probabilidad. Este valor se asocia a una etiqueta clasificatoria o *"label"*. Esta probabilidad final (para cada etiqueta) es la predicción inicial de que la imagen (con la que inició el proceso) pertenezca a una etiqueta.

La función de activación softmax es bastante especial. Si esta función de activación fuese otra, tendríamos resultados que no podrían ser usados como probabilidades. Podríamos tener una etiqueta con una probabilidad de 65 % y otra con probabilidad de 50 %. Una imagen debe de, por fuerza, pertenecer a una de las clases o etiquetas, esto significa que una imagen que entre a la red debe tener una probabilidad de que pertenece a una clase. Como se habla de probabilidades, entonces la suma de los resultados de cada predicción tienen que ser igual a 1.

En la primer iteración de la red, lo más probable es que la primer predicción indique que la primer imagen pertenece a una clase a la que no pertenece. Por ejemplo, si la primer imagen es la de un gato, lo más probable que suceda es que la red dictamine que la imagen pertenezca a otra etiqueta, como podría ser la de peces.

En este momento comienza el aprendizaje. Para que la red arroje diferentes predicciones que se acerquen a la correcta, la red debe **corregir y aprender de sus errores**.

Es importante tomar en cuenta que las CNN entran en lo que se conoce como entrenamiento supervisado. Lo que sucede es que sabiendo a qué etiqueta pertenece una imagen (100 % de probabilidad de que la imagen de un gato pertenezca a la clase 'gato'), se tomará la predicción original de la red (supongamos que la predicción inicial arrojó que la imagen inicial tiene un 20 % de probabilidad de pertenecer a la clase 'gato') y se calculará el error entre la predicción y la realidad. Muchos autores simplemente usan la palabra error, e intentan explicar el cálculo usando la definición del error cuadrado medio como método de cálculo de error, sin embargo, lo que sucede en realidad es que lo que se usa para asignar probabilidades y calcular error es la suma de la función softmax con otra función llamada *"categorical cross-entropy"*.

Para entender mejor como trabaja esta función, debemos entender que es la entropía. La entropía es la cantidad de información (en bits) que posee un evento relacionado con la incertidumbre de dicho evento. Matemáticamente:

$$I(a) = -\log_2 P(a), \tag{2.12}$$

donde $P(a)$ es la probabilidad de que el evento a ocurra, e $I(a)$ es la información que dicho evento aporta.

Sin embargo, es de interés conocer la información que aporta no sólo un evento, sino una serie de eventos probabilísticos. La entropía de una variable aleatoria X se define como el valor esperado de información que acarrea un resultado de X. El valor esperado ($E[X]$) de una variable aleatoria es el promedio de todos los valores de dicha variable,

multiplicados por su probabilidad (Kulkarni, 2019). La entropía H de una distribución de probabilidad P de una variable aleatoria se define como:

$$H(P) = E_{X\sim P}[-\log_2 P(X)], \tag{2.13}$$

donde E es el valor esperado de la distribución de probabilidad, X es la variable aleatoria y $X \sim P$ significa que los valores que toma X son de la distribución P (Kulkarni, 2019).

Ahora bien, cuando se habla de entropía cruzada, nos referimos a la entropía relativa entre dos distribuciones de probabilidad medida sobre el mismo conjunto de eventos. Para obtener la entropía cruzada entre las distribuciones P y Q, se calcula la entropía de Q usando las probabilidades de P. De manera matemática, esta entropía se observa en la Ecuación 2.14 (Kulkarni, 2019; DaneriCoding, 2017; Wang et al., 2019).

$$H(P, Q) = E_{X\sim P}[-\log_2 Q(X)] = -\sum_i P(X_i) \log_2 Q(X_i) \tag{2.14}$$

Ahora bien, *"categorical cross-entropy"* se utiliza como una función de pérdida cuando se trabaja en *"deep learning"*. Cuando una CNN necesita clasificar una imagen entre un número de etiquetas, utiliza la distribución de probabilidad de las etiquetas $y(x)$ (es decir, cada probabilidad de que una etiqueta pertenezca a cada una de las etiquetas clasificatorias) y la distribución de probabilidad de etiquetas predicha $y'(x)$ que otorga la función softmax y calcula la entropía cruzada entre ambas distribuciones, como se observa en la expresión 2.15.

$$H(y, y') = -\sum_i y(x_i) \log_2 (y'(x_i)), \tag{2.15}$$

donde y_i es la distribución real obtenida a partir del etiquetado de los datos y y'_i es la distribución de probabilidad generada por nuestro modelo por la función softmax.

La función se conoce como *"binary cross-entropy"* cuando solo son dos clases, y se usa la expresión *"categorical cross-entropy"* cuando se usan más de dos clases.

"Categorical cross-entropy" es nuestra función de pérdida o costo (no error cuadrado

medio u otras) y es la expresión que se busca minimizar. Esto ocurre cuando las dos distribuciones de probabilidad son la misma.

$$\begin{gathered} si\ y' \sim y \\ H(y', y) = H(y, y) \to 0 \end{gathered} \tag{2.16}$$

Una vez que este error se haya calculado, la CNN, mediante un concepto avanzado denominado *"Backpropagation"* (retropropagación, en español), retornará este error a cada etapa, modificando los pesos y *"bias"* de cada tramo hasta el comienzo, con el objetivo de que cuando la imagen del gato vuelva a viajar a través de la red, el proceso matemático de convolución y multiplicación de pesos arroje un resultado tal que la probabilidad de que la imagen pertenezca a la clase 'gato' sea más grande. Este proceso de *"Backpropagation"* contiene cálculo de gradientes y derivadas parciales. El gradiente se calcula con las derivadas parciales de la función de costo a la salida con respecto únicamente a los parámetros de la última capa, es decir los últimos pesos antes de ese punto.

Matemáticamente, la derivada parcial de la función de costo con respecto a los últimos pesos w no puede calcularse directamente, así que se utiliza la regla de la cadena para este cálculo, como se expresa:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L}, \tag{2.17}$$

donde C es la función de pérdida o coste, que para nuestra explicación es la función *categorical cross-entropy*; a^L es la función de activación en esta capa (de ahí el superíndice L, de capa, en inglés), la cual en este caso es la función softmax; z^L es la suma ponderada conformada por la multiplicación de los valores de la capa anterior por los pesos de la capa actual (de ahí el superíndice L) y finalmente w^L son los pesos de la capa actual.

Una vez obtenidas estas primeras derivadas, pasamos a la capa anterior, y se obtienen nuevamente las derivadas parciales de la función costo pero ahora con respecto a los parámetros de esta capa (de ahí el superíndice $L-1$), con ayuda de la regla de la cadena.

Basándonos en la expresión anterior, el siguiente paso se expresaría de la siguiente forma:

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial w^{L-1}}, \tag{2.18}$$

Esto continúa hasta el comienzo de la red neuronal. De esta manera propagamos hacía atrás el error, de tal forma que la red aprende.

Cada iteración de la red hará que esta se vuelva más y más precisa hasta que sea capaz de identificar la imagen inicial con alta precisión. El proceso continúa afinando su conocimiento con imágenes de la misma clase (otros gatos, por ejemplo) y los pesos se alinearán para identificar las características de la etiqueta 'gatos' en general. En este mismo proceso de entrenamiento entrarán también imágenes de otras clases, digamos 'peces'. El proceso de *"Backpropagation"* se encargará de modificar los pesos de manera que estas imágenes sean identificadas correctamente sin alterar tanto el aprendizaje que ya tenga sobre otras clases.

Con este proceso arduo y complejo la red neuronal gana un aprendizaje que puede ser utilizado para poder clasificar imágenes fuera del grupo con el que entrenó.

2.2.10. Parámetros de Aprendizaje

Es importante aterrizar una CNN en la realidad, es decir, recordar que una CNN es un programa. Se debe diseñar, programar, parametrizar, y sobre todo ejecutar. En la CNN los metaparámetros pueden estar reconfigurándose de acuerdo a los resultados obtenidos.

Uno de estos metaparámetros configurables es el optimizador. La optimización es el camino que el *"Backpropagation"* puede seguir. Matemáticamente existen muchos algoritmos que toman el error calculado por la función softmax y el *"cross entropy categorization"* y lo usan para corregir los pesos de manera recursiva.

El optimizador básico se denomina descenso de gradiente estocástico (SGD). Al utilizar

este optimizador, se actualiza la matriz de pesos W de la siguiente manera:

$$w_{k+1} = w_k - \alpha \frac{\partial C}{\partial w}, \tag{2.19}$$

donde α denota *"learning rate"* o tasa de aprendizaje que regularmente es 0.01 (Chollet et al., 2015a), $\frac{\partial C}{\partial w}$ denota el gradiente de la matriz de pesos, la cual es el resultado del *"Backpropagation"* y k es el índice de la muestra.

El optimizador Adam es el optimizador más utilizado en la actualidad debido a que presenta resultados óptimos en problemas de *"deep learning"* (Kingma and Ba, 2014). En este se calcula la media (m) y la varianza (v) de los gradientes (momentos probabilísticos, de ahí el nombre). En este, el peso se actualiza con base a estos parámetros de acuerdo a las siguientes ecuaciones:

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{\frac{v_k}{1-\beta_2}} + \epsilon} \times \frac{m_k}{1-\beta_1}, \tag{2.20}$$

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\left(\frac{\partial C}{\partial w}\right), \tag{2.21}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2)\left(\frac{\partial C}{\partial w_k}\right)^2, \tag{2.22}$$

donde $\beta_1 = 0.9$, $\beta_2 = 0.999$, k es el índice de iteración y $\epsilon = 10^{-8}$ (Kingma and Ba, 2014; Chollet et al., 2015b). Este último parámetro es una constante que se usa para evitar la división entre 0.

2.2.11. Entrenamiento, Validación y Prueba

Las CNN deben pasar por etapas de entrenamiento, validación y prueba. Para poder ser entrenadas, estas redes necesitan como entrada conjuntos de imágenes etiquetadas.

- Entrenamiento: Son las imágenes utilizadas para ajustar los parámetros de la red. Estas imágenes son las que pasan por los filtros para que la red aprenda sus carac-

terísticas.

- Validación: Se usan después de cada fase del proceso de entrenamiento para comprobar si no hay *"overfitting"* o *"underfitting"*.
- Prueba: Se utilizan después de completar el entrenamiento para probar el acierto de la red.

Para entrenar una red neuronal la mejor expresión que lo describe es 'entre más, mejor'. Regularmente se entrena con un conjunto de imágenes grande. No existe un número 'óptimo', ni máximo ni mínimo: lo único que se puede decir es que entre más grande sea la cantidad de imágenes, mejor es la calidad del entrenamiento. En redes neuronales actuales, se usan conjuntos de imágenes de cantidades de al menos tres cifras, sin embargo esta solo es una regla que se sugiere en la literatura y su éxito es empírico.

Lo que se hace es que se dividen las imágenes en dos subconjuntos: entrenamiento y prueba. El porcentaje de la división varía pero lo que es seguro es que el conjunto de imágenes de entrenamiento debe ser superior en cantidad que el conjunto de imágenes de validación. Las imágenes que se destinaron a la parte de entrenamiento se vuelven a dividir. Este subconjunto se vuelve un segundo 100 %, y se divide nuevamente. De manera semejante, se destina un porcentaje mayor de esta división al entrenamiento y el porcentaje de imágenes menor se destina a las pruebas (Torres, 2019).

2.2.12. *"Overfitting"* y *"Underfitting"*

Los errores más comunes que pueden presentarse al momento de entrenar la red son el *"overfitting"* y el *"underfitting"*. Como se mencionó antes, estos no son errores referentes al código o ni referentes al proceso matemático, sino son eventos que indican una incorrecta predicción.

El modelo presenta *"underfitting"* (subajuste) cuando los valores de exactitud y pérdida se quedan estancados. Esto quiere decir que la red no tiene suficientes muestras, por lo

que no puede generalizar el conocimiento y por lo tanto, no arroja una buena predicción.

Por el contrario, cuando el modelo tiene *"overfitting"* (sobreajuste) el entrenamiento es bueno, pero la validación no presenta ninguna mejora. Cuando pasa esto es porque la red aprendió tantos parámetros en el entrenamiento, que ahora solo reconoce las imágenes utilizadas para entrenar, y no puede predecir nuevas imágenes.

En la Figura 2.9 se ilustra la manera en que los resultados de una CNN se manifiestan cuando existe *"overfitting"*, el funcionamiento óptimo (*"perfect fitting"*) y cuando hay *"underfitting"*.

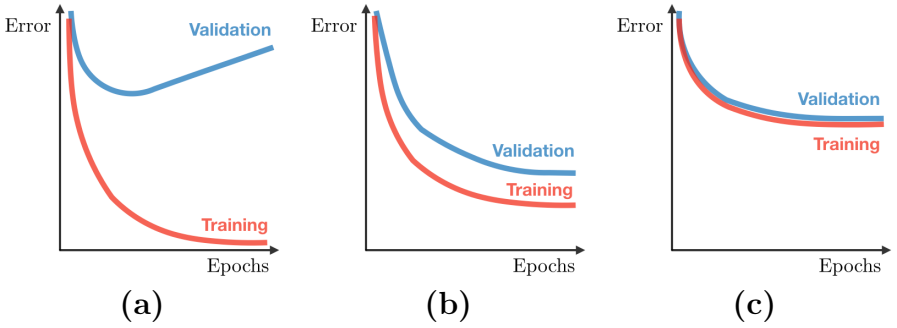


Figura 2.9: Gráficas de resultados posibles de una CNN: **(a)** *"Underfitting"*, **(b)** Resultado óptimo, **(c)** *"Overfitting"* (Amidi and Amidi, 2018).

2.2.13. Tasa de Aprendizaje

Se ha comentado anteriormente que una CNN aprende modificando los valores de los pesos de las diferentes capas mediante la propagación del error hacia atrás. Estos valores de pesos no pueden ser calculados de manera analítica, y es por eso que la red opera de manera 'empírica', corrigiendo sus errores. Cuando una red se corrige así misma en cada iteración, se corrige con un paso al que se conoce como *"learning rate"* o tasa de aprendizaje, siendo uno de los hiperparámetros más importantes que se tienen que sintonizar en una CNN. El *"learning rate"* es un número positivo pequeño, que se encuentra entre 0 y 1. Este hiperparámetro no posee unidad.

$$1 > \alpha > 0 \tag{2.23}$$

Capítulo 3

Diseño y Parámetros de la Red Neuronal

3.1. Introducción

Este capítulo presenta información detallada acerca de la programación y el diseño de la CNN. Se detalla información de las bibliotecas necesarias para la correcta implementación de la CNN. Se explica también información acerca de las imágenes que se utilizaron para este trabajo. Lo más importante es que se analiza la estructura de la red, explícandola en sus múltiples etapas.

En la práctica, una CNN se diseña, se programa, se codifica y se experimenta en el ambiente de programación de elección. Los dos ambientes más populares son `Anaconda` de Anaconda Inc y `Google Collab` de Google. Redes creadas en estos ambientes se codifican en el lenguaje de programación Python.

El diseño de CNN que se presenta en este capítulo tiene como objetivo la estimación discreta (clasificación) de los flujos másicos de glicerina y aire que fluyen simultáneamente en una tubería a partir de mediciones de presión. Para el entrenamiento de la red se emplearon imágenes de espectrogramas que se obtuvieron con la transformada de tiempo corto

de Fourier (STFT es su acrónimo en inglés derivado de: *"short-time Fourier transform"*), la cual se aplicó a los datos que midieron cuatro transductores de presión instalados a lo largo de una tubería de flujo bifásico alojada en el Instituto de Ingeniería de la UNAM. Mayores detalles de la experimentación se dan en el Apéndice A.

3.1.1. TensorFlow

`Tensorflow` es una biblioteca de código abierto para aprendizaje automático, desarrollado por Google para satisfacer las necesidades de sistemas capaces de construir y entrenar redes neuronales que detecten y descifren patrones y correlaciones, análogamente al aprendizaje y razonamiento usado por los humanos.

3.1.2. Keras

`Keras` es una API (conjunto de funciones e instrucciones que pueden ser utilizadas como interfaz simplificatoria entre un software más complicado y el usuario) de alto nivel diseñada para la creación y entrenamiento de modelos de *"deep learning"*. `Keras` está escrita en Python y usa `Tensorflow` como *"backend"*, es decir que realiza las funciones mientras que `Tensorflow` las operaciones y procesos matemáticos. Fue diseñado para ser amigable con el usuario, es modular y extensible.

3.2. Imágenes

Como ya se mencionó, el fin de esta tesis y, en particular, de este capítulo es el diseño de una red neuronal entrenada con imágenes para que sea utilizada como herramienta en aplicaciones (industriales o académicas) que involucren flujo bifásico glicerina-aire. Las preguntas que surgen al diseñar una CNN: ¿qué imágenes usar?, ¿cuántas imágenes son convenientes para entrenar?, etc. Para el entrenamiento de la CNN desarrollada en esta tesis, se utilizaron espectrogramas que se obtuvieron aplicando la STFT a las series de

tiempo de presión proporcionadas por cuatro transductores instalados a lo largo de una tubería de flujo bifásico alojada en el Laboratorio de Hidrodinámica del Instituto de Ingeniería (Hernández et al., 2019). La STFT se implementó en MATLAB. Un ejemplo de estos espectrogramas que se utilizan en el entrenamiento de la red neuronal se ilustra en la Figura 3.1.

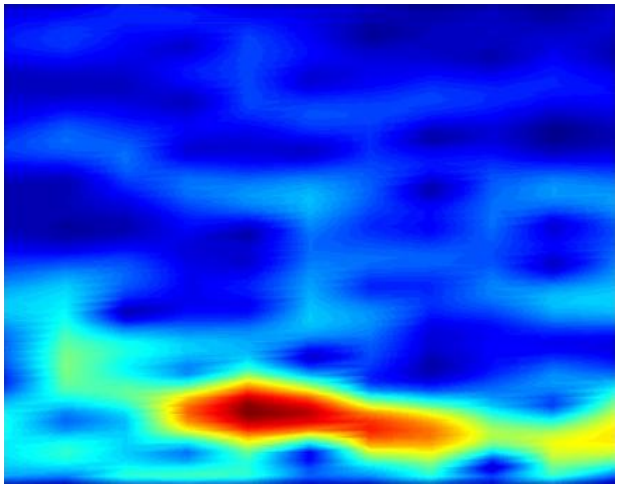


Figura 3.1: Imagen usada en el entrenamiento, obtenido durante el experimento denominado 1. Por la tubería corrían 0.005 [kg/s] de aire y 1.3 [kg/s] de glicerina. Con respecto a los espectrogramas, el eje x representa el tiempo [s], el eje y representa la frecuencia [Hz] y el eje z o color representa el diferencial de presión [kPa].

Al inicio de este proyecto se contaba con datos de presión para generar apenas 90 imágenes, las cuales se encuentran clasificadas de acuerdo a varios parámetros: el experimento donde se obtuvieron, el gasto másico de glicerina, el gasto másico de aire (medido en kg/s, pero etiquetado en g para diferenciarlo del flujo másico de glicerina), y el diferencial de presión que da origen a la imagen. En la Figura 3.1 se observa el espectrograma llamado **Exp1_1.3kg/s_5g_dP1.jpg**. Esta imagen se obtuvo del experimento 1, cuando fluían 1.3 kg/s de glicerina y 0.005 kg/s de aire. Los datos de esta imagen se tomaron restando los valores del sensor de presión 1 (P1) al segundo sensor (P2), es decir obteniendo el diferencial de presión entre la posición de P1 y la posición de P2, el cual se etiquetó dP1. Con 4 sensores de presión instalados, se obtuvieron tres diferenciales de presión por experimento a partir de los cuales se generaron los espectrogramas.

Una CNN requiere para su entrenamiento una elevada cantidad de imágenes que ga-

rantice su correcto funcionamiento y obtención de resultados útiles, como lo sería obtener predicciones con exactitud del 90 %. El entrenamiento de una CNN requiere la afinación de metapárametros, afinación del diseño, correcciones, y un sinfín de ediciones. Sin embargo, una condición necesaria para entrenar una CNN es una cantidad de imágenes considerable, y como se aconseja en la literatura (no hay una fórmula, ni nada exacto para decir cuantas imágenes son necesarias para entrenar una CNN) se requiere una cantidad de imágenes por cada etiqueta clasificatoria de por lo menos 4 cifras ("Tensorflow". CODEVSTACK. Comunicación personal. Enero, 13 al 17, 2020).

Para el entrenamiento, una CNN necesita que el conjunto de imágenes de trabajo estén debidamente clasificadas. La clasificación es bastante sencilla. Para clasificar imágenes simplemente basta con que estén dentro de una carpeta con el nombre de la clase. La CNN trabajó con diversas clases.

Cuando se clasificó sólo con respecto al gasto másico del aire, se trabajó solamente con tres clases: `5g/s, 10g/s, y 15g/s.` Es decir, todas las imágenes se separaron manualmente en estas 3 carpetas, de acuerdo a su gasto másico de aire, sin considerar el otro gasto másico u otros identificadores (el experimento, el diferencial de presión o la modificación a la imagen).

Cuando se clasificó solo con respecto al gasto másico de la glicerina, se trabajó con cinco clases: `1.3kg/s, 2.5kg/s, 3.7kg/s, 4.9kg/s y 6.1kg/s.` Todas las imágenes se separaron manualmente en estas 5 carpetas, de acuerdo a su gasto másico de glicerina, sin considerar el otro gasto másico u otros identificadores.

Finalmente, cuando se clasificó con respecto a ambos gastos másicos, se trabajó con la máxima cantidad de clases, es decir quince: `1.3kg/s_5g/s, 2.5kg/s_5g/s, 3.7kg/s_5g/s, 4.9kg/s_5g/s, 6.1kg/s_5g/s, 1.3kg/s_10g/s, 2.5kg/s_10g/s, 3.7kg/s_10g/s, 4.9kg/s_10g/s, 6.1kg/s_10g/s, 1.3kg/s_15g/s, 2.5kg/s_15g/s, 3.7kg/s_15g/s, 4.9kg/s_15g/s y 6.1kg/s_15g/s.` Todas las imágenes se separaron manualmente en estas 15 carpetas, de acuerdo a sus dos gastos másicos al mismo tiempo pero sin considerar otros identificadores.

3.2.1. Aumento de Imágenes

Evidentemente, las pocas imágenes con las que se contó al comienzo fueron insuficientes, así que se solicitaron más datos. Los datos otorgados de otros experimentos realizados alzaron la cantidad de imágenes a 279. Este número de imágenes sigue siendo muy pequeño en contraste con el usado regularmente en otras redes neuronales, por tanto se buscaron soluciones a este problema, y se encontró una técnica denomina aumento de imágenes (*"image augmentation"*) (Torres, 2019, Capítulo 6).

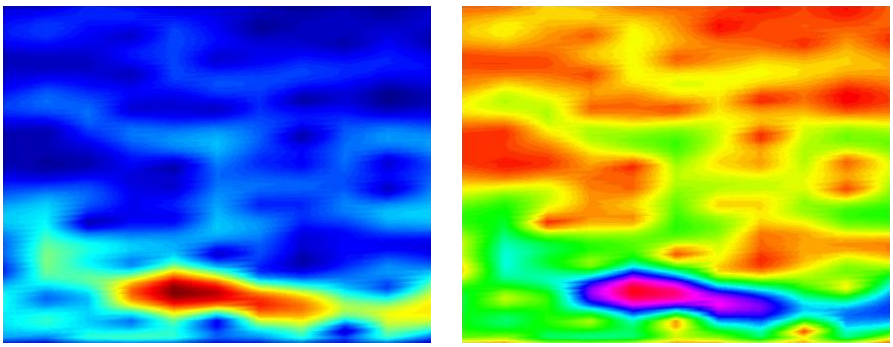
El aumento de imágenes es un tipo de aumento de datos, *"Data augmentation"*, el cual es un proceso externo al entrenamiento de las CNN cuyo único objetivo es el de aumentar la cantidad de datos que se tienen para entrenar la red cuando estos son insuficientes. Este proceso se realiza con anterioridad al entrenamiento de la CNN y no forma parte del programa. De esta manera se tiene una cantidad de imágenes superior previas antes de todo. Asegurarse de tener una buena cantidad de imágenes es el paso 0 para entrenar una CNN. El aumento de imágenes se refiere a modificar múltiples aspectos de una imagen para así multiplicarlas y aumentar la piscina de imágenes totales. Para este caso, se utilizó `ImageDataGenerator`, la cual es una clase de `Keras` que permite hacer el aumento de imágenes. Existen muchas modificaciones que puede realizar esta clase de `Keras`, pero las utilizadas fueron las siguientes:

1. Cambio de Canal. Vuelve más blanca la imagen.
2. Espejo Vertical. Voltea la imagen sobre el eje vertical.
3. Rotación Aleatoria. Rota la imagen en un rango aleatorio. Puede ser rotación horaria o antihoraria.
4. Remarcado. Remarca figuras de la imagen.
5. Zoom Aleatorio. Realiza cierto zoom sobre alguna parte de la imagen.
6. Oscurecido. Oscurece la imagen.

7. Movimiento. Mueve la imagen hacia una dirección: a) Derecha, b) Izquierda, c) Arriba y d) Abajo.

En la Figura 3.3 se aprecia la imagen original y el uso de aumento de imágenes para multiplicarla por 10.

Para obtener aun más imágenes, se procedió a transformar las imágenes originales a través de MATLAB, cambiando su paleta de colores. Las imágenes originales usaban la paleta predeterminada JET, y se optó por cambiarla por la paleta HSV. El contraste entre ambas paletas se aprecia en Figura 3.2.



(a) **(b)**

Figura 3.2: Cambio de paleta de color. **(a)** Imagen con colores de la paleta JET de Matlab. **(b)** Imagen con colores de la paleta HSV de Matlab.

Una vez obtenidas las imágenes con el color cambiado, se realizó el aumento de imágenes y se pueden observar ejemplos en la Figura 3.4.

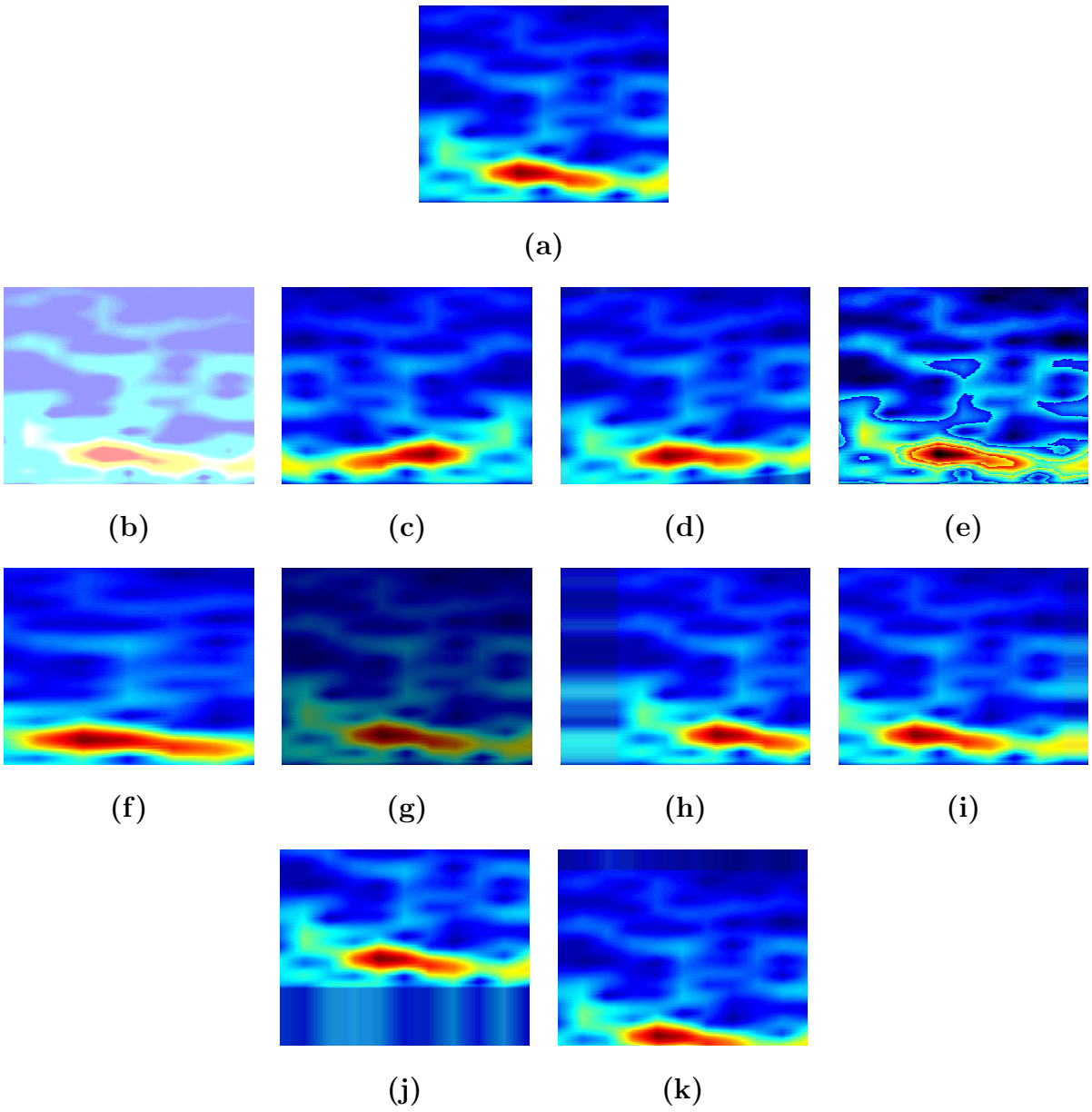


Figura 3.3: Aumento de imágenes, colores de la paleta JET de Matlab: **(a)** Imagen Original. **(b)** Imagen aclarada. **(c)** Imagen volteada sobre el eje vertical. **(d)** Imagen ligeramente rotada. **(e)** Imagen remarcada. **(f)** Imagen con zoom. **(g)** Imagen oscurecida. **(h)** Imagen arrastrada a la derecha. **(i)** Imagen arrastrada a la izquierda. **(j)** Imagen arrastrada hacia arriba. **(k)** Imagen arrastrada hacia abajo.

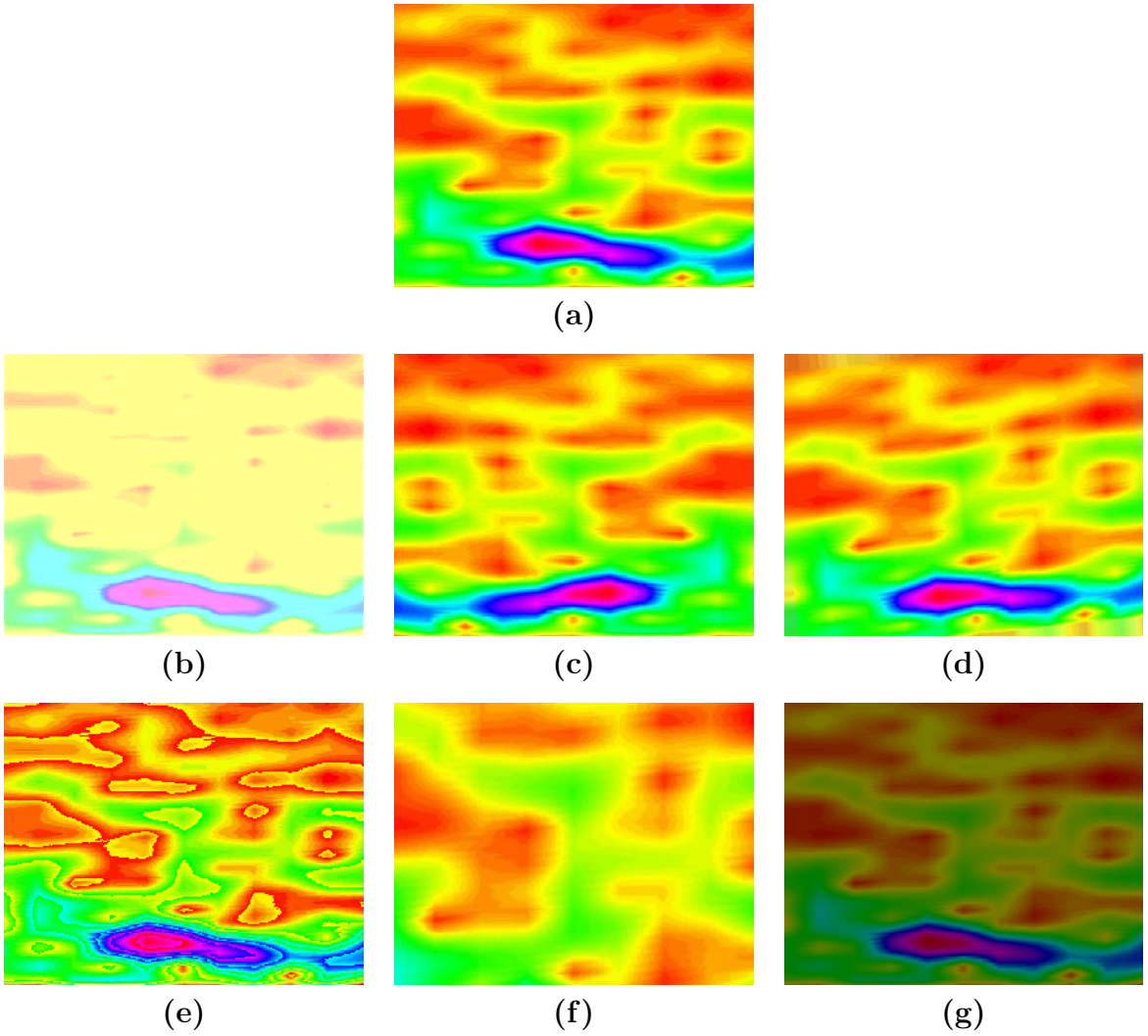


Figura 3.4: Aumento de imágenes, colores de la paleta HSV de Matlab: **(a)** Imagen Original. **(b)** Imagen aclarada. **(c)** Imagen volteada sobre el eje vertical. **(d)** Imagen ligeramente rotada. **(e)** Imagen remarcada. **(f)** Imagen con zoom. **(g)** Imagen oscurecida.

3.3. Estructura de la Red Neuronal Convolucional

La CNN de este proyecto se programó en Anaconda y se encuentra basada en la CNN de Bagnato (2018). Después de innumerables ediciones se calibró un modelo con un porcentaje de predicción aceptable y este programa se puede analizar a detalle en el Anexo B.

En la Figura 3.5, se observan los pasos que sigue la red neuronal al efectuar su proceso. Cada etapa del proceso se explicará en las subsecciones siguientes.

3.3.1. Etapa 1. Importación de Bibliotecas

Se mencionan aquí las distintas bibliotecas necesarias para el correcto funcionamiento de la CNN:

(a) `TensorFlow`. Biblioteca de código abierto que puede relacionar varios datos simultáneamente, para construir y entrenar redes neuronales. En esta red es utilizado como el *"backend"* de `Keras`.

(b) `Keras`. Es la API que funge como interfaz entre `Tensorflow` y el programador, permitiendo que el entorno sea más amigable y el código menos extenso.

(c) `Numpy`. Significa `Numerica Python`. Es una biblioteca que permite potentes cálculos con matrices multidimensionales, además de contar con herramientas para trabajar con esas matrices.

(d) `Matplotlib.pyplot`. Provee una manera de graficar semejante a MATLAB. Está principalmente diseñada para generar gráficas interactivas.

(e) `Os` (módulo). Este módulo provee una manera portable de usar funcionalidades dependientes del sistema operativo. Sirve para abrir, cerrar o leer archivos, manipular rutas, crear archivos temporales y direcciones entre otras utilidades. Particularmente, en esta red, tiene la función de obtener la ruta de la carpeta en la que se encuentran las imágenes.

Figura 3.5: Estructura de la Red Neuronal Convolucional.

(f) `Re` (módulo). Este módulo sirve para buscar coincidencias con un patrón de búsqueda. Particularmente en esta red se utiliza para encontrar las imágenes que se utilizarán para entrenar.

(g) `Scikit-learn`. Es una biblioteca utilizada para el aprendizaje de máquina. Incluye varios algoritmos populares de aprendizaje automático, como regresión y clasificación.

(h) `Scikit-image`. Contiene una colección de algoritmos para procesamiento de imágenes.

3.3.2. Etapa 2. Cargado de Imágenes

Por razones de simplicidad, la explicación sobre el desarrollo de la CNN se hará mencionando solamente tres etiquetas referentes al gasto másico de aire. Sin embargo, el trabajo se realizó con más etiquetas o variaciones de las mismas, como lo son las 5 etiquetas de gasto másico de glicerina solamente, o la combinación de ambos gastos másicos resultando en 15 etiquetas, e innumerables variaciones de esos intentos al sintonizar múltiples metapárametros. En el capítulo 4 Resultados de la Red Neuronal se pueden observar los resultados obtenidos con diversos experimentos referentes a la cantidad de etiquetas empleadas.

Una vez que las bibliotecas se han cargado, el programa de entrenamiento de la CNN busca la carpeta en donde se tienen las imágenes. Estas carpetas estarán divididas en subcarpetas que representan cada clase que queremos predecir. Estas imágenes se cargan a la red.

En esta misma sección, se modifica el tamaño de las imágenes, a 56×56 píxeles en este caso. El tamaño original de las imágenes es de 276×217 píxeles, sin embargo, pueden entrar imágenes de cualquier tamaño. Este recorte es debido a que si las imágenes se dejaran en su tamaño real se utilizarían muchas más neuronas y volvería el proceso de aprendizaje mucho más complejo, lo cual ocuparía recursos que no se poseen. El cambio de tamaño de cada imagen se realiza inmediatamente se carga.

3.3.3. Etapa 3. Asignación de Etiquetas

A cada imagen se le asigna como etiqueta el nombre de la carpeta en la que se encuentra. Además, se convierten las imágenes en matrices de píxeles utilizando la librería `numpy`. En una variable de tipo matriz se colocan las matrices de píxeles de todas las imágenes y en otra variable, de tipo arreglo, se guardan las etiquetas de cada imagen, como se ilustra en la Figura 3.6.

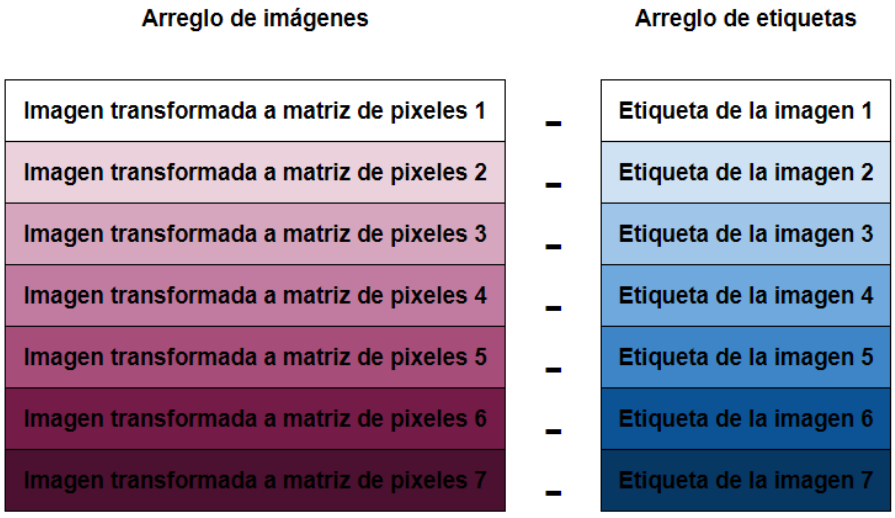


Figura 3.6: Se emparejan 2 arreglos. Uno contiene todas las imágenes transformadas a matrices de píxeles y el otro tiene las respectivas etiquetas de cada imagen. Cada imagen se relaciona con su etiqueta.

Se le asigna una clase a cada etiqueta. Eso significa que se le dará un número del 0 al 2 a cada carpeta, por que se está trabajando solo con 3 clases. Esto con la finalidad de que cuando se le pida a la red identificar una imagen, esta nos diga el número al que corresponde, como se muestra a continuación:

$$\begin{aligned} 0 &\rightarrow 10\ (\text{g/s}) \\ 1 &\rightarrow 15\ (\text{g/s}) \\ 2 &\rightarrow 5\ (\text{g/s}) \end{aligned}$$

3.3.4. Etapa 4. Normalización de Imágenes y Etiquetas

En esta etapa, se asigna el número de imágenes que se utilizarán en el entrenamiento, en las pruebas y en la validación. De manera general, la división quedó de la siguiente manera: se asignó 72.25 % del total de las imágenes al entrenamiento, 12.75 % del total de las imágenes a la validación y el 15 % restante a la prueba, como se ilustra en la Figura 3.7.

Al comienzo se asignó 85 % del total de las imágenes a un subconjunto inicial de entrenamiento y el 15 % restante a la prueba. Los arreglos de entrenamiento se guardan en las variables `train_image` (que contiene las imágenes) y `train_label` (que contiene las etiquetas). Por otro lado, los arreglos de pruebas se guardarán en `test_image` y `test_label`. Las variables `train_image` y `test_image` se normalizarán al dividirse entre 255. Se divide entre este número porque el valor de la variable es un número entre 0 y 255, correspondiente al valor de un píxel, pero para nuestra red, nos conviene que sea un valor entre 0 y 1.

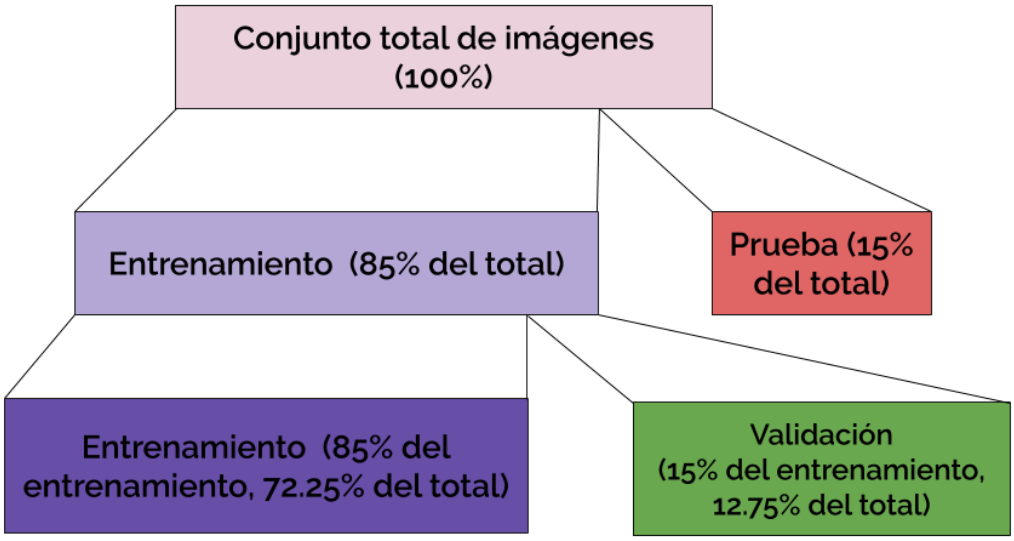


Figura 3.7: Asignación de las imágenes a cada subconjunto necesario para el proceso: Entrenamiento, Prueba y Validación

A las variables `train_label` y `test_label` se les aplicará un protocolo denominado *"one-hot encoding"*, porque así es más fácil clasificar para la red. *"One-hot encoding"* puede entenderse como la asignación de un valor con un número compuesto por ceros y unos, donde el número es la posición del 1 con respecto a los ceros. Con el *"one-hot encoding"*, las etiquetas quedarán de la siguiente manera:

$$\begin{aligned} 0 &\rightarrow 100 \\ 1 &\rightarrow 010 \\ 2 &\rightarrow 001 \end{aligned}$$

Posteriormente, del subconjunto de imágenes de entrenamiento inicial se generan dos subconjuntos: de este 100 % se asignaron 85 % a un subconjunto de entrenamiento y el 15 % restante a la validación. En la Figura 3.8 se ilustra el orden en que suceden las separaciones, la normalización y el *"one-hot encoding"*.

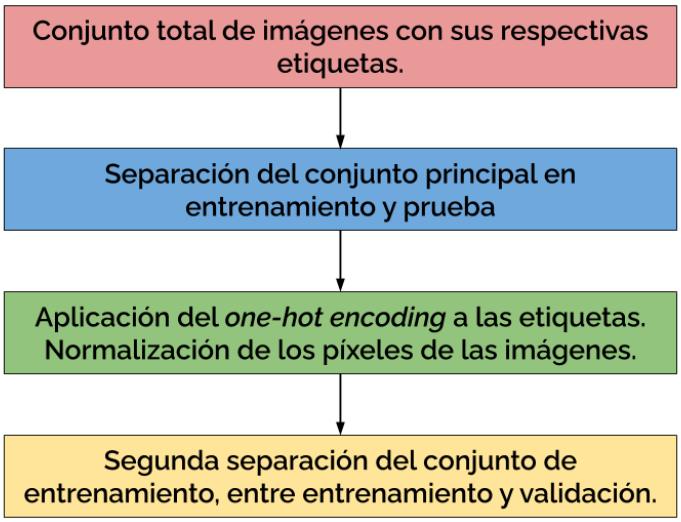


Figura 3.8: Orden en que suceden las separaciones de los conjuntos y la posición del *"one-hot encoding"* y la normalización de los píxeles en este proceso.

3.3.5. Etapa 5. Diseño del Modelo

A continuación se describen las características principales del modelo de la CNN.

Para comenzar, el modelo se define como secuencial. Esta red posee solamente una capa convolucional. Esta capa está conformada por 32 *"kernels"*, cada uno de tamaño 3×3. La capa tiene *"padding"* en su configuración de *"same"*, es decir está conformado de ceros. El *"stride"* de esa capa es de 2×2. La función de activación utilizada en esta capa es ReLU. Todos estos metaparámetros se eligieron de manera heurística. A través de la comparación de resultados de múltiples variaciones de la red fue que se llegó a estas configuraciones.

La capa siguiente es la capa de *"max pooling"*, la cual es de tamaño 3×3. También posee *"padding"* igualmente en configuración *"same"*.

Para un mejor desempeño de la CNN, se optó por utilizar una capa de *"dropout"*. Esta inhabilita solo el 25 % de los parámetros que pasan por ella.

Después se encuentra una capa de *"batch normalization"*. Y posteriormente se colocó la capa *"flatten"*. Luego, sigue la primer capa densamente conectada. Esta capa cuenta con 32 neuronas y su función de activación es Softmax. Posteriormente, se colocó nuevamente una capa de *"dropout"*, la cual inhabilita el 30 % de los parámetros que la atraviesan. Finalmente, la última capa es la capa densamente conectada de salida, cuyo número de neuronas simplemente depende de la cantidad de clases. Se trabajó con 3, 5 y 15 neuronas en diversos experimentos. La función de activación de esta capa final es la función Softmax.

La configuración final es la relativa compilación del modelo como tal. En esta, se calibran la función de pérdida y el optimizador. Para este modelo, la función de pérdida utilizada fue *"categorical cross entropy"*. El optimizador seleccionado fue el optimizador Adam, porque como se mencionó en la subsección Parámetros de Aprendizaje, es un optimizador muy eficaz. Después de repetidas pruebas, se llegó a la conclusión de que la tasa de aprendizaje de dicho optimizador se mantuviera en $\alpha = 0.0002$, porque es el valor que presentó mejores resultados.

Tabla 3.1: Resumen del Modelo

Tipo de Capa	Tamaño de Salida	Parámetros
Convolucional	$28 \times 28 \times 32$	896
"Max Pooling"	$10 \times 10 \times 32$	0
"Dropout"	$10 \times 10 \times 32$	0
Normalización de lote (BN)	$10 \times 10 \times 32$	128
"Flatten"	3200	0
Densamente Conectada	32	102432
"Dropout"	32	0
Densamente Conectada	3	99

3.3.6. Etapa 6. Entrenamiento de la Red

En esta etapa se realizó el entrenamiento de la red, es decir, se dejó ejecutando el programa, definiendo el metaparámetro *"epochs"*, el cual es el número de iteraciones que la CNN va a ejecutar. Para poder analizar el comportamiento del entrenamiento, esta variable tomó valores desde 10 hasta 200, con un valor regular de 100.

En este proceso se mostraron las pérdidas y la exactitud de cada iteración de la CNN mientras corría.

La red entrenó con las imágenes y etiquetas de entrenamiento, y mientras se ejecutaba, el sistema iba validando con las imágenes y etiquetas destinadas a la validación.

Al terminar el entrenamiento, se guardó el modelo para no tener que entrenarla otra vez en el futuro. Este modelo entrenado puede usarse más adelante.

3.3.7. Etapa 7. Evaluación de la Red

En esta etapa se realizó un examen a la red. Se lleva a cabo utilizando las imágenes que fueron reservadas exclusivamente para dicha prueba. Dado que estas imágenes no se usaron en el entrenamiento, la CNN no las conoce, por lo que es capaz de probar si las reconoce con lo aprendido en el modelo. La prueba o examen nos arroja resultados de la exactitud (*"accuracy"*) y las pérdidas (*"loss"*) del modelo.

3.3.8. Etapa 8. Generación de Predicción

Para esta etapa se generó un nuevo programa. Su objetivo es cargar el modelo previamente guardado y luego cargar una imagen completamente nueva para que pase por el mismo proceso. Esta imagen no debe estar en los subconjuntos de Entrenamiento y Validación, y tiene la finalidad de ilustrar que la red funciona, y que es capaz de clasificarla en alguna etiqueta. La imagen se reajusta a un tamaño de 56×56, se convierte en matriz, se normaliza, atraviesa el proceso y finalmente observamos en que etiqueta se colocó.

Este programa se hizo para poder hacer uso del guardado del modelo de la red, y no tener que entrenarla cada vez que se desee utilizar.

Capítulo 4

Resultados de la Red Neuronal

Este capítulo detalla el desempeño que tuvo la red neuronal durante el proceso de diseño y calibración, y se muestran los diversos resultados que se obtuvieron con el aumento progresivo del número de imágenes para el entrenamiento.

Estos resultados se plasman de dos formas distintas para las tres etapas: como exactitud y pérdida en la fase de entrenamiento, como exactitud y pérdida con el conjunto de imágenes de validación y finalmente como exactitud y pérdida en la sección final de prueba. Para la validación y el entrenamiento, los resultados se encuentran en forma de gráficas de exactitud y pérdida (o costo) medidas contra las iteraciones. Para la parte de prueba el programa entrega de manera directa los valores de exactitud y pérdida.

La exactitud se mide en porcentaje y la pérdida es adimensional. Los puntos azules en las gráficas de exactitud representan el acierto en el entrenamiento y la línea roja representa el acierto en la validación. Funciona de manera semejante en las gráficas de pérdida. Los puntos azules representan el valor final de pérdida en el entrenamiento, y la línea roja representa el valor final de la pérdida en la validación.

Esta gráficas son utilizadas como indicador del desempeño de las CNN. Una CNN tiene un comportamiento correcto si sus gráficas se comportan de la siguiente manera:

1. Las gráficas de exactitud (en entrenamiento y validación) deben **subir** como una exponencial positiva, y su comportamiento es especialmente bueno si son semejantes, paralelas y, aún mejor, si están una sobre la otra.

2. Las gráficas de pérdida (en entrenamiento y validación) deben **bajar** como una exponencial negativa, y su comportamiento es especialmente bueno si son semejantes, paralelas y, todavía mejor, si están una sobre la otra.

Las métricas se interpretan de la siguiente forma:

- La pérdida en el entrenamiento es el error con el que trabaja la CNN (es decir, el error que se va minimizando en el proceso de *"Backpropagation"*) cuando compara su propio aprendizaje (generado con el conjunto de imágenes destinadas al entrenamiento) contra las imágenes que generaron ese aprendizaje.

- La pérdida en la validación es el error que se genera cuando se compara la predicción del modelo entrenado con el conjunto de imágenes destinadas a la validación.

Conforme avanzan las iteraciones, ambos errores deben ser menores y es por eso que ambas gráficas de pérdida deben converger a un valor pequeño.

- La pérdida en la prueba es el error que se genera cuando se compara la predicción del modelo entrenado con el conjunto de imágenes destinadas a la prueba. Pero en esta etapa final, las imágenes no atraviesan por iteraciones, si no que solo se comparan con el modelo una sola vez, de tal forma la pérdida es un resultado singular no graficable.

- La exactitud en el entrenamiento es el porcentaje de acierto que se obtiene al comparar las etiquetas predichas para el conjunto de imágenes de entrenamiento y las etiquetas verdaderas para cada imagen. Es la comparación entre lo que la CNN es capaz de predecir contra las etiquetas que generaron ese mismo aprendizaje.

- La exactitud en la validación es semejante a la del entrenamiento, salvo que la comparativa de etiquetas predicha y verdaderas sucede con imágenes del conjunto de validación (ninguna presente en el entrenamiento), donde la predicción se genera con el aprendizaje generado en la etapa de entrenamiento.

La exactitud en ambos tipos de gráficas está medido en porcentaje. Un 0 % de exactitud nos habla de una CNN que no pudo aprender nada. Una exactitud del 100 % nos habla de una CNN que es capaz de predecir de manera óptima.

Conforme avanzan las iteraciones, el aprendizaje debe ser capaz de identificar mejor cada imagen del conjunto de validación y entrenamiento, entonces estas exactitudes deben ser mayores y es por eso que ambas gráficas de exactitud deben subir.

- Finalmente, la exactitud en la prueba es semejante a las exactitudes anteriores, salvo que la comparativa de etiquetas predicha y verdaderas sucede con imágenes del conjunto de prueba (ninguna presente en el entrenamiento), donde la predicción se genera con el aprendizaje generado en la etapa de entrenamiento. Al igual que con su contraparte, la pérdida, las imágenes no atraviesan por iteraciones, si no que solo se comparan con el modelo una sola vez, de tal forma la exactitud es un resultado singular no graficable.

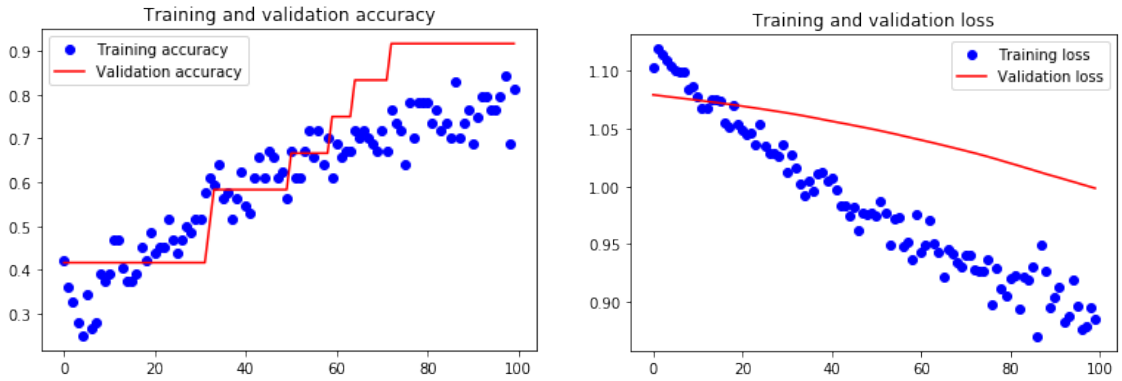
El proceso para el entrenamiento de la CNN propuesta en esta tesis se describe a continuación.

Primeramente, se entrenó una CNN con 90 imágenes para predecir el gasto de aire. Los índices de desempeño de la CNN fueron bajos (menores al 60 %). Después, se entrenó una CNN con 279 imágenes para predecir el gasto másico de aire. El desempeño de la CNN no mejoró con respecto a la previamente entrenada. Posteriormente, se entrenó una CNN para predecir ambos gastos másicos, tanto el de aire como el de glicerina. Para el entrenamiento se utilizaron 279 imágenes originales, las cuales se modificaron para ser utilizadas en el entrenamiento, es decir, se llevó a cabo el proceso de *"data augmentation"* que se explicó

en el capítulo 3. El desempeño de la red continuó siendo bajo. Finalmente, se entrenó un modelo doble, el primero para predecir el gasto másico de aire y el segundo para predecir el gasto másico de glicerina. El entrenamiento se realizó con las 279 imágenes originales y modificaciones de ellas. El desempeño de predicción de esta CNN mejoró notablemente.

4.1. Resultados con Noventa Imágenes

Originalmente se inició el proyecto con una cantidad de 90 imágenes. Estas fueron probadas con un prototipo de red que pudiera identificar el gasto másico de aire en la tubería (cuando se hicieron los experimentos que dieron lugar a las imágenes), es decir, podía identificar entre 3 etiquetas: 5 g/s, 10 g/s y 15 g/s. El desempeño en las etapas de entrenamiento y validación se ilustra en las gráficas de la Figura 4.1.



(a) **(b)**

Figura 4.1: Resultados obtenidos con 90 imágenes. **(a)** Exactitud. **(b)** Pérdida.

Al momento de usar el conjunto de imágenes de prueba, se obtuvo una exactitud máxima del 57 % y una pérdida de 1.053. Lo que se aprecia es que la exactitud no es muy alta y la pérdida deja de descender y se estanca. La red presentó un *"underfitting"*, debido a la poca cantidad de imágenes usadas.

4.2. Resultados con Doscientas Setenta y Nueve Imágenes

Posteriormente, el número de datos otorgados por el Instituto de Ingeniería aumentó ya que se realizaron nuevos experimentos, con lo cual se pudieron generar nuevas imágenes. El número de imágenes para el entrenamiento de la red aumentó a un total de 279. De ese total, a 131 imágenes se les asignó la etiqueta de 5 g/s, a 75 la etiqueta de 10 g/s y finalmente a 75 la de 15g/s. El entrenamiento del modelo con dicha cantidad de imágenes arrojó los resultados de entrenamiento y validación plasmados en la Figura 4.2.

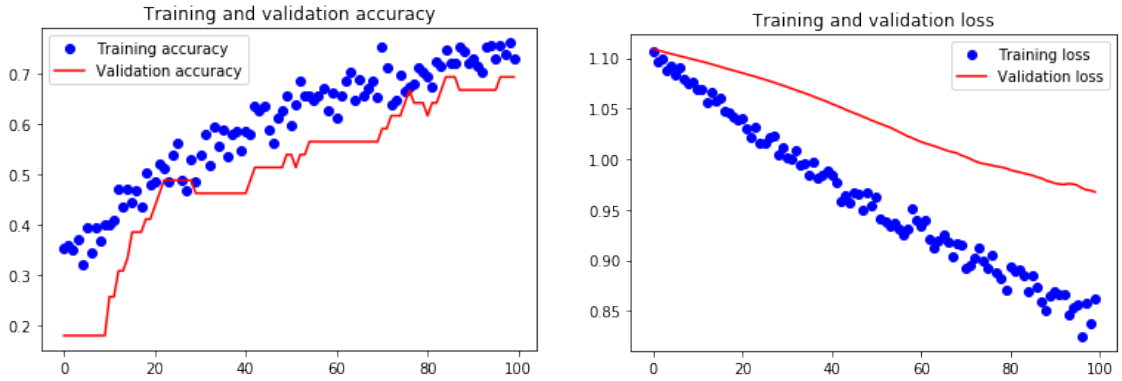


Figura 4.2: Resultados obtenidas con 279 imágenes. **(a)** Exactitud. **(b)** Pérdida.

Se puede observar con respecto a las figuras anteriores que no hubo gran mejora. Se sospecha que esto se debe a que, a pesar de la semejanza de los experimentos, las condiciones de la tubería cambiaron durante la experimentación, e.g, cambios en la temperatura ambiente. Puede que esta sea la razón por la que las imágenes no sean tan parecidas y por tanto puede ser una razón por la que la CNN no sea tan eficaz generalizando el conocimiento.

De igual forma se realizaron pruebas con este modelo y se obtuvieron resultados del 47 % de exactitud y una pérdida de 1.077.

4.3. Modelo para Predecir Aire y Glicerina

Como se mencionó en el capítulo anterior, la gran falta de imágenes llevó a la generación de imágenes nuevas a partir de modificaciones de las originales. Recordando el capítulo 3 de este trabajo, el método para la creación de imágenes nuevas a partir de ligeras modificaciones en ellas es conocido como *"image augmentation"*, el cual es un tipo de *"data augmentation"*, utilizados para aumentar los datos en un experimento de *"machine learnig"* cuando estos datos son insuficientes para obtener resultados favorables. Para esta prueba se dividieron las imágenes en 15 conjuntos y a cada conjunto se le adjudicó una etiqueta. La cantidad de imágenes asignadas a cada etiqueta se puede apreciar en la Tabla 4.1.

Tabla 4.1: Cantidad de Imágenes por Etiqueta

Etiqueta	Número de Imágenes por etiqueta
5g/s_1.3kg/s	594
5g/s_2.5kg/s	270
5g/s_3.7kg/s	595
5g/s_4.9kg/s	270
5g/s_6.1kg/s	594
10g/s_1.3kg/s	262
10g/s_2.5kg/s	258
10g/s_3.7kg/s	262
10g/s_4.9kg/s	258
10g/s_6.1kg/s	258
15g/s_1.3kg/s	270
15g/s_2.5kg/s	258
15g/s_3.7kg/s	258
15g/s_4.9kg/s	258
15g/s_6.1kg/s	258

El entrenamiento del modelo con dicha cantidad de imágenes y etiquetas arrojó los resultados de validación y entrenamiento plasmados en la Figura 4.3.

Cabe mencionar que desde el punto de vista gráfico, en las figuras se puede apreciar que las gráficas de entrenamiento y validación se comportan de manera similar, y toman una forma aceptable.

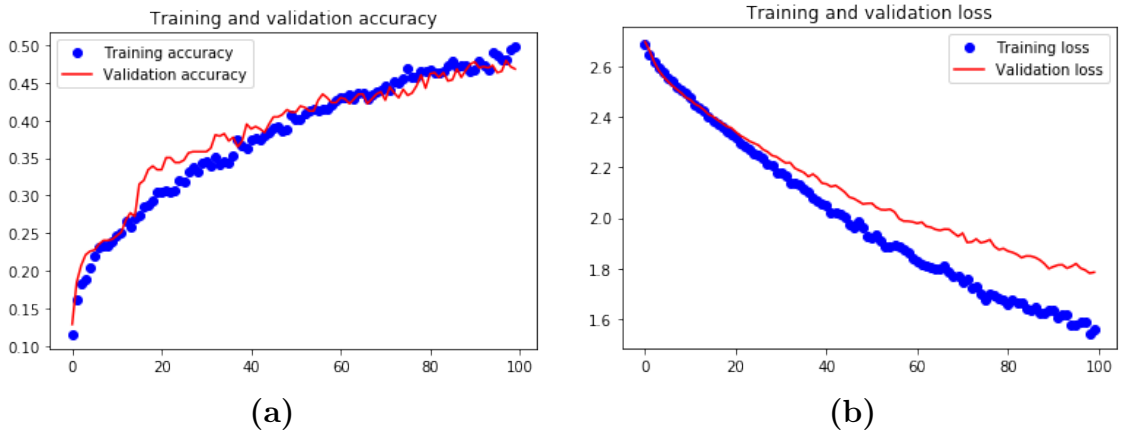


Figura 4.3: Resultados obtenidos con *"data augmentation"* y 15 etiquetas. **(a)** Exactitud. **(b)** Pérdida.

El problema surgió al probar la red con el conjunto de imágenes destinadas a la prueba, los resultados de exactitud y pérdida no mejoraron mucho. Esto se debe a la alta cantidad de etiquetas que se tienen, puesto que las imágenes por carpeta son insuficientes y esto minimiza el desempeño de la red. Cada etiqueta con la que se trabaje en una CNN debe tener una considerable cantidad de imágenes asociadas.

Este modelo arrojó resultados de 48 % de exactitud y una pérdida de 1.809. Del conjunto de imágenes de prueba (739 en total), la red pudo acertar correctamente la etiqueta de 336 imágenes y erró en 403.

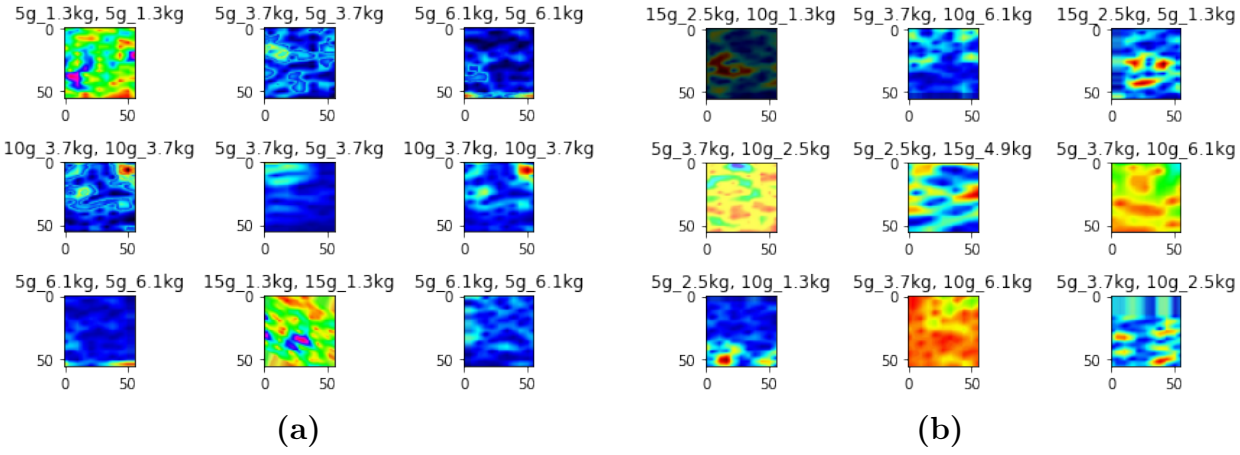


Figura 4.4: Clasificación de imágenes por la red: **(a)** Ejemplos de imágenes correctamente etiquetadas. **(b)** Ejemplos de imágenes incorrectamente etiquetadas.

En la Figura 4.4 se ilustran imágenes clasificadas por la CNN. De cada una de las subfiguras, el texto en la esquina izquierda es la etiqueta predicha por la red, y el texto de la esquina derecha es la etiqueta asociada con la subfigura. De tal forma que se puede apreciar ejemplos de imágenes correctamente e incorrectamente catalogadas.

4.4. Propuesta de Modelo Doble

En las subsecciones anteriores se exploraron diversas maneras de entrenar la CNN y se aumentó la cantidad de imágenes de entrenamiento. Sin embargo, a pesar del aumento de imágenes, el desempeño de la CNN no fue satisfactorio. Esto se debe a que el desempeño de la CNN disminuye al separar las imágenes en un gran número de etiquetas. Para resolver esto, se propuso lo siguiente: en vez de que un solo modelo entrenado asigne 15 etiquetas a las imágenes, es mejor que se entrenen dos modelos, el primero para identificar (predecir) el gasto másico de glicerina (es decir las etiquetas 1.3 kg/s, 2.5 kg/s, 3.7 kg/s, 4.9 kg/s, 6.1 kg/s) y el segundo para identificar el gasto másico de aire (es decir, las etiquetas 5 g/s, 10 g/s, y 15 g/s). Se realizaron pruebas usando la más alta cantidad de imágenes disponibles usando las nuevas cantidades de etiquetas, y se encontraron los resultados descritos a continuación.

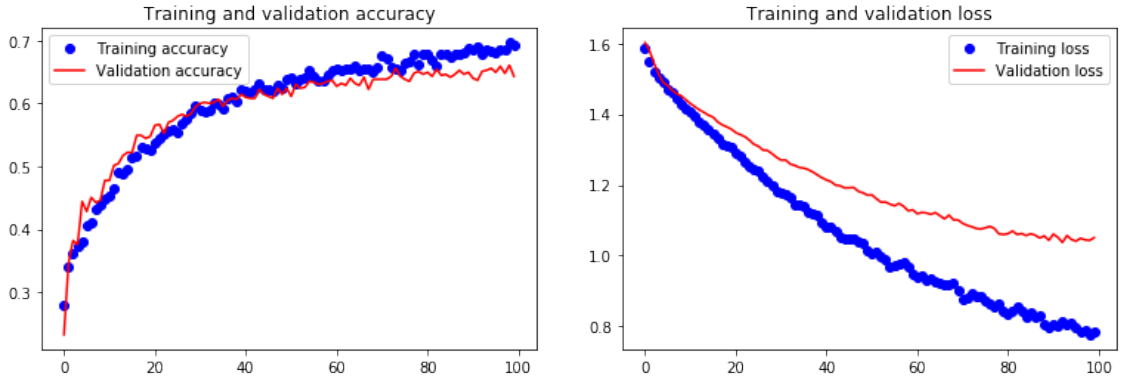
4.4.1. Modelo de Gasto Másico de Glicerina

La nueva separación de imágenes totales y su etiqueta asignada se aprecia en la Tabla 4.2.

A través del uso exclusivo de las etiquetas de la tabla 4.2 que identifican el gasto de glicerina, se puede apreciar en el entrenamiento y la validación una considerable mejora, la cual se aprecia en las gráficas de la Figura 4.5.

Tabla 4.2: Cantidad de Imágenes por Etiqueta

Etiqueta	Número de Imágenes por etiqueta
1.3 kg/s	1126
2.5 kg/s	786
3.7 kg/s	1115
4.9 kg/s	786
6.1 kg/s	1110



(a) **(b)**

Figura 4.5: Resultados obtenidas en este experimento: **(a)** Exactitud, **(b)** Pérdida.

Las gráficas de entrenamiento y validación se comportan de manera similar, y toman una forma bastante correcta. Ambas pérdidas igualmente se comportan bastante acorde con lo esperado.

Desde el punto de vista analítico, los resultados de exactitud y pérdida mejoraron considerablemente. Este modelo arrojó, en la etapa de prueba, resultados de 68 % de exactitud y una pérdida de 0.9982, los cuales fueron los mejores resultados obtenidos hasta ese momento. Del conjunto de imágenes de prueba (739 en total), la red pudo acertar correctamente la etiqueta de 503 imágenes y erró en 236. En la Figura 4.6 se ejemplifican imágenes clasificadas por la CNN correctamente e incorrectamente catalogadas.

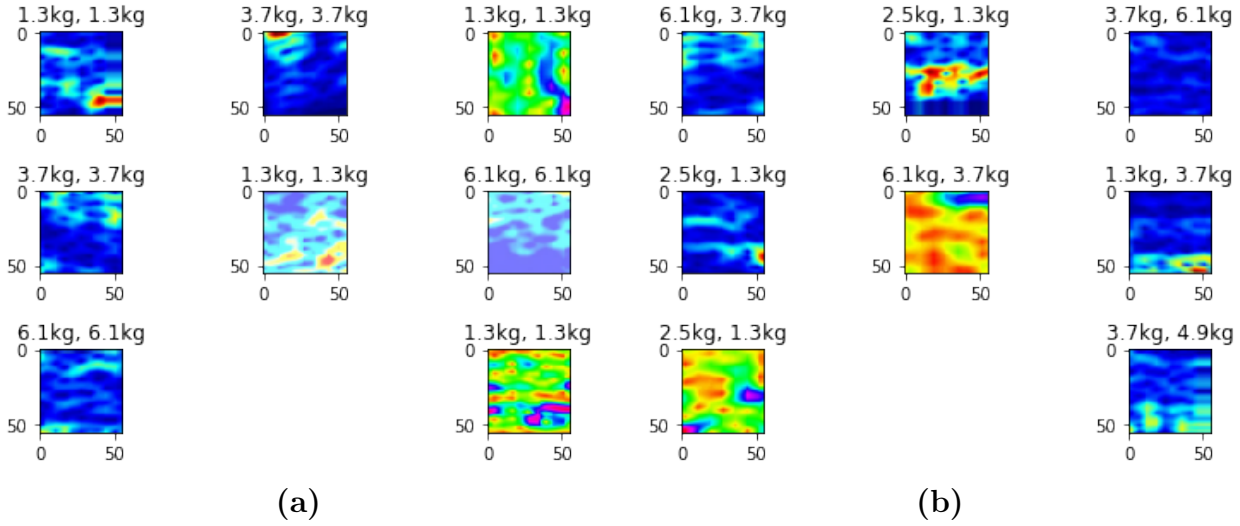


Figura 4.6: Clasificación de imágenes por la red. **(a)** Ejemplos de imágenes correctamente etiquetadas. **(b)** Ejemplos de imágenes incorrectamente etiquetadas.

4.4.2. Modelo de Gasto Másico de Aire

La segunda nueva separación de imágenes totales con su etiqueta asignada se aprecia en la Tabla 4.3.

Tabla 4.3: Cantidad de Imágenes por Etiqueta

Etiqueta	Número de Imágenes por etiqueta
5 g/s	1274
10 g/s	1176
15 g/s	1176

A través del uso exclusivo de las etiquetas anteriores que identifican gasto másico de aire, se puede apreciar en el entrenamiento y la validación una considerable mejora, la cual se observa en las gráficas de la Figura 4.7.

Las gráficas de entrenamiento y validación se comportan de manera similar, y toman una forma bastante correcta. Ambas pérdidas igualmente se comportan bastante acorde con lo esperado.

Desde el punto de vista analítico, los resultados de exactitud y pérdida alcanzaron los

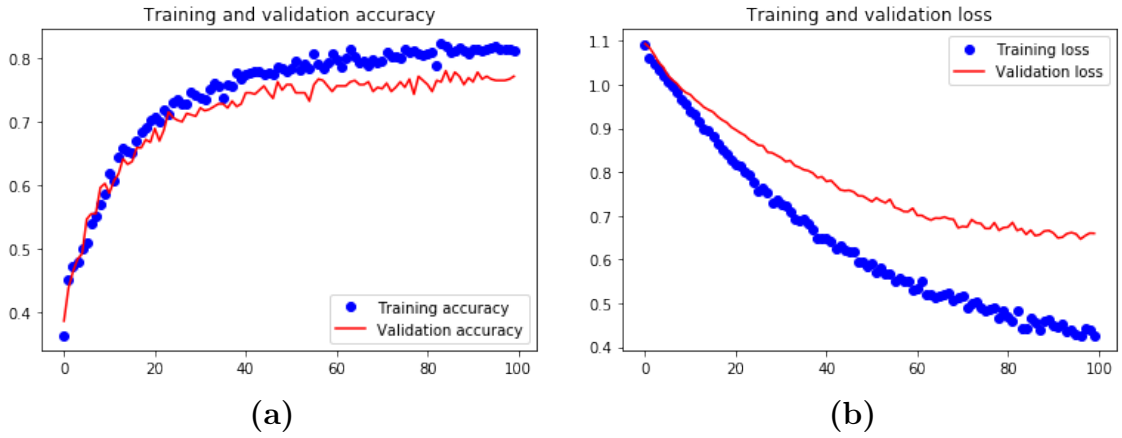


Figura 4.7: Resultados obtenidos en este experimento. **(a)** Exactitud. **(b)** Pérdida.

mayores valores vistos. Este modelo arrojó resultados, para etapa de prueba, de 78 % de exactitud y una pérdida de 0.6185, los cuales fueron los mejores resultados obtenidos de todos. Del conjunto de imágenes de prueba (544 en total), la red pudo acertar correctamente la etiqueta de 428 imágenes y erró en 116. En la Figura 4.8 se ejemplifican imágenes clasificadas por la CNN correctamente e incorrectamente catalogadas.

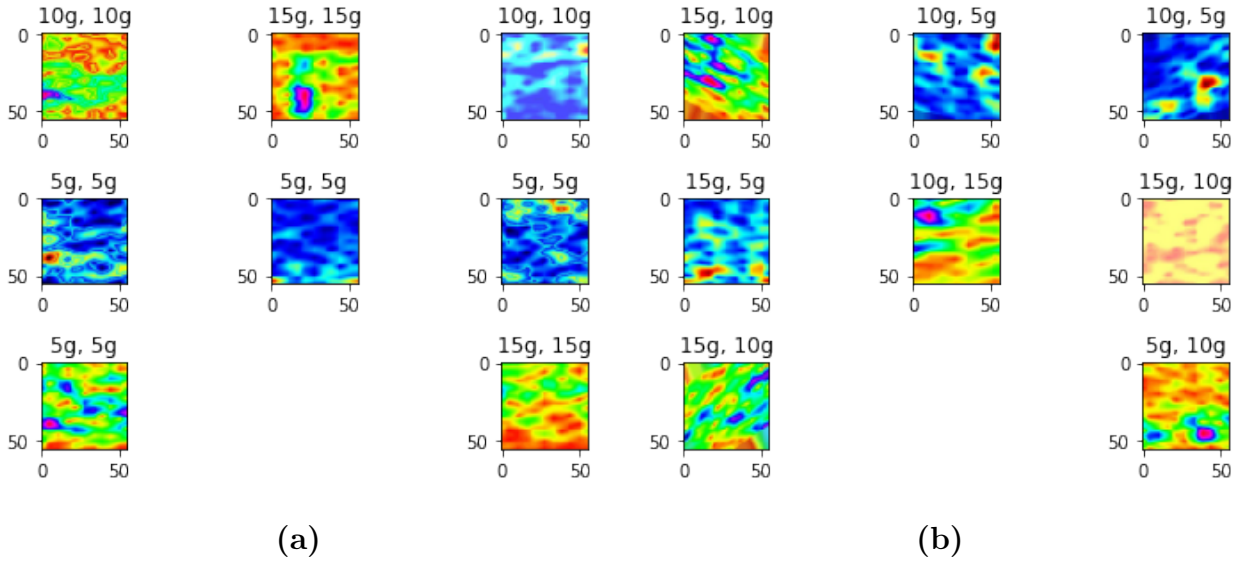


Figura 4.8: Clasificación de imágenes por la red: **(a)** Ejemplos de imágenes correctamente etiquetadas, **(b)** Ejemplos de imágenes incorrectamente etiquetadas.

4.5. Efectos del Uso de *"Dropout"*

En la subsección 2.2.6 *"Dropout"* se comenta que se decidió dejar las capas de *"dropout"* por que su uso presentaba mejores resultados en general. A continuación se presentan los resultados obtenidos cuando se entrena la CNN utilizando las mismas condiciones del apartado anterior (subsección 4.4.2. Modelo de Gasto Másico de aire) pero con la diferencia de que se retiró las capas *"dropout"* del código. Los resultados de entrenamiento y la validación se observan en las gráficas de la Figura 4.9, donde se aprecia que hay más *"overfitting"*.

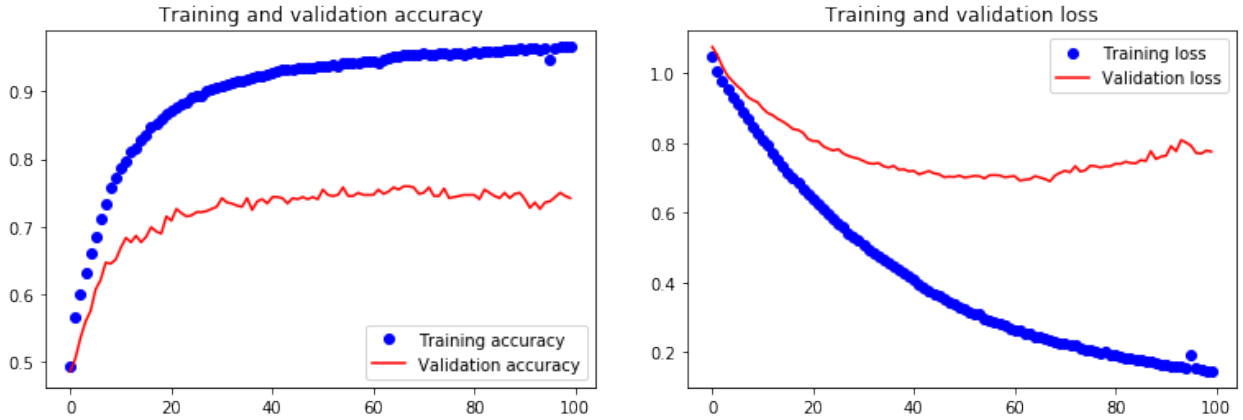


Figura 4.9: Resultados obtenidos con la totalidad de las imágenes disponibles y 3 etiquetas clasificatorias pero sin la capa *"dropout"*. **(a)** Exactitud. **(b)** Pérdida.

Los resultados de exactitud y pérdida que este modelo arrojó para la etapa de prueba fueron de 72 % de exactitud y una pérdida de 0.728. Esta exactitud es menor a la del apartado anterior, donde la exactitud fue del 78 % y la pérdida fue mayor a la obtenida anteriormente, la cual fue de 0.6185. Si comparamos las gráficas de la Figura 4.7 con las de la Figura 4.9 se observa que las gráficas que no poseen *"dropout"* divergen.

4.6. Predicción de Cantidad de Imágenes Necesarias

Como se mencionó en secciones anteriores, una cantidad considerable de imágenes es necesaria para entrenar una CNN tal que esta nos otorgue resultados aceptables. La pregunta inevitable es: ¿cuántas imágenes son requeridas para asegurar un resultado aceptable? Pues, como se mencionó en el capítulo 2, no existe una fórmula que nos indique la cantidad precisa que se necesite para cumplir con este objetivo. Sin embargo, de manera empírica se sabe que se necesitan 'miles' de imágenes.

Estas aseveraciones llevaron a la búsqueda de un número estimado de imágenes necesarias, basado en el éxito que tuvieron las imágenes que se poseen. Para lograr este estimado, se utilizaron regresiones lineales. Como se sabe, el proceso de regresión lineal necesita un conjunto de pares ordenados como entrada, y a la salida nos da parámetros con los que se puede construir la ecuación de una recta, la cual es la recta que más se amolda a los datos de entrada.

Para este caso, se busca obtener el número de imágenes promedio por etiqueta que se requiere para obtener los mejores resultados desde el punto de vista de la exactitud y desde el punto de vista de la pérdida. Entonces, los pares ordenados son la cantidad de imágenes promedio por etiqueta de los múltiples experimentos (3 etiquetas, 5 etiquetas y 15 etiquetas) y los resultados que se obtuvieron con dichos promedios por etiqueta, es decir la exactitud y la pérdida obtenidas. En las Tablas 4.4, 4.5 y 4.6 se muestran los puntos utilizados en las regresiones lineales.

Tabla 4.4: Tres Etiquetas Clasificatorias

Número de Imágenes Promedio por Etiqueta	Exactitud [%]	Pérdida
30	57	1.0532
93.66	47	1.077
1208.66	78	1.809

Cada uno de los renglones de estas tres tablas resume un experimento realizado con la CNN. Por ejemplo, el primer renglón de la Tabla 4.4 resume uno de los primeros experi-

Tabla 4.5: Cinco Etiquetas Clasificatorias

Número de Imágenes Promedio por Etiqueta	Exactitud [%]	Pérdida
18	42	1.5626
55.8	42	1.7419
984.6	65	0.9825

Tabla 4.6: Quince Etiquetas Clasificatorias

Número de Imágenes Promedio por Etiqueta	Exactitud [%]	Pérdida
6	7	2.6979
23	23	2.55
327.4	48	1.809

mentos que se llevaron acabo con la CNN, cuando solo se poseían apenas 90 imágenes y se trabajó solo con 3 etiquetas, resultando en un promedio de 30 imágenes por etiqueta. Con esas pocas imágenes se obtuvo apenas un 57 % de exactitud y una pérdida de 1.0532. Los demás renglones funcionan de manera similar. Conforme se fueron obteniendo más imágenes, los resultados mejoraron, lo cual puede interpretarse como una recta, y estas rectas se pueden obtener con los pares ordenados mostrados en la tablas a través de simple regresión lineal. La cantidad de imágenes por etiquetas forman la variable independiente x y los resultados las variables dependientes Acc (exactitud) y $Loss$ (pérdida).

Los resultados de las regresiones lineales con los pares ordenados permitieron la construcción de las expresiones presentadas en las Ecuaciones 4.1- 4.6. Cada una de estas rectas se grafican en la Figura 4.10.

Exactitud con 3 Etiquetas Clasificatorias.

$$Acc_3 = 0.0223x + 50.7822 \tag{4.1}$$

Pérdida con 3 Etiquetas Clasificatorias.

$$Loss_3 = -3.8766 \times 10^{-4}x + 1.0884 \tag{4.2}$$

Exactitud con 5 Etiquetas Clasificatorias.

$$Acc_5 = 0.0242x + 41.1147 \tag{4.3}$$

Pérdida con 5 Etiquetas Clasificatorias.

$$Loss_5 = -5.6645 \times 10^{-4}x + 1.5388 \tag{4.4}$$

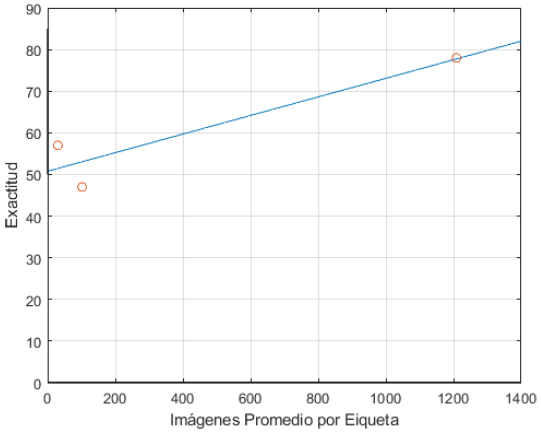
Exactitud con 15 Etiquetas Clasificatorias.

$$Acc_{15} = 0.1062x + 13.5046 \tag{4.5}$$

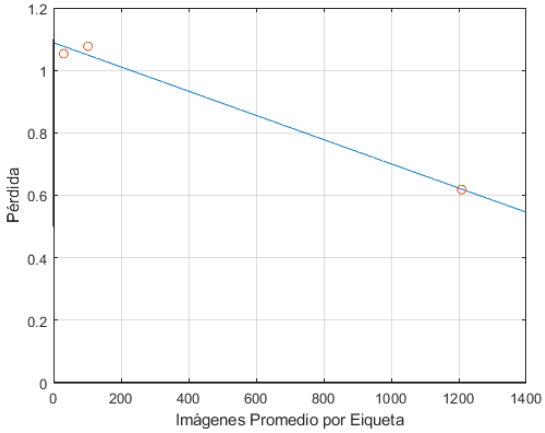
Pérdida con 15 Etiquetas Clasificatorias.

$$Loss_{15} = -26 \times 10^{-4}x + 2.6583 \tag{4.6}$$

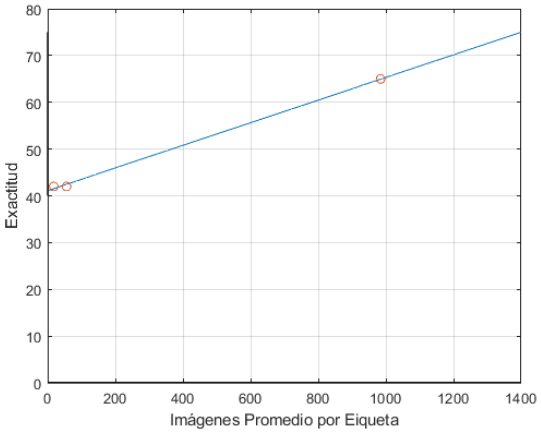
Las primeras 2 gráficas de la Figura 4.10 explican el desempeño de la CNN desde el punto de vista de la exactitud y la pérdida en las pruebas cuando se trabajó con 3 etiquetas clasificatorias (5 g/s, 10 g/s y 15 g/s). Los primeros puntos ilustran la exactitud y pérdida para cuando solo un promedio de 30 imágenes por etiqueta estaban asignadas a cada clasificación. Los segundos puntos ilustran la exactitud y pérdida para cuando había, en promedio, 93 imágenes por etiqueta asignadas a cada clasificación. Finalmente los terceros puntos ilustran la exactitud y pérdida para cuando la máxima cantidad de imágenes disponibles se asignaron, es decir, un promedio de 1208 imágenes por etiqueta estaban asignadas a cada clasificación. Las 2 gráficas siguientes ilustran lo mismo, pero cuando se trabajó con 5 etiquetas clasificatorias (1.3 kg/s, 2.5 kg/s, 3.7 kg/s, 4.9 kg/s y 6.1 kg/s). Finalmente las 2 gráficas finales ilustran lo anterior, solo que con la máxima cantidad de etiquetas (la combinación de etiquetas clasificatorias de flujo másico liquido y gaseoso) es decir 15 etiquetas.



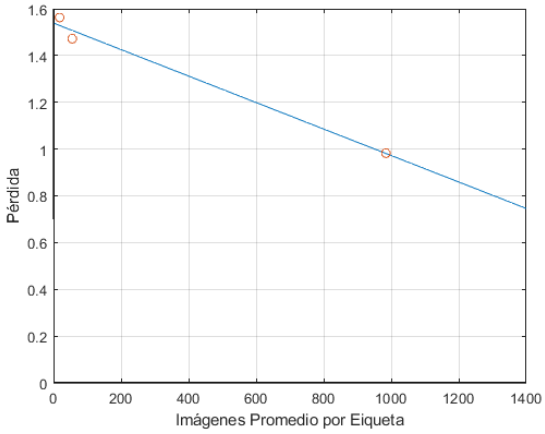
Exactitud. 3 Etiquetas



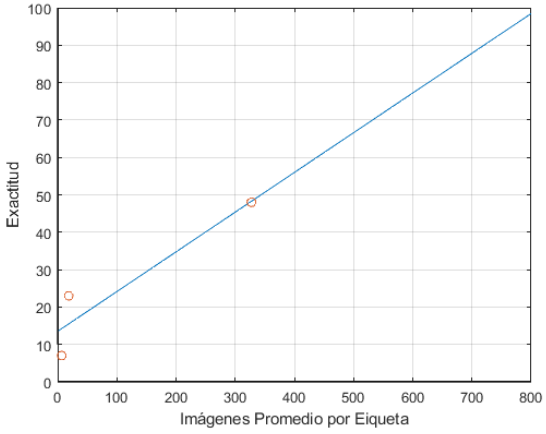
Pérdida. 3 Etiquetas



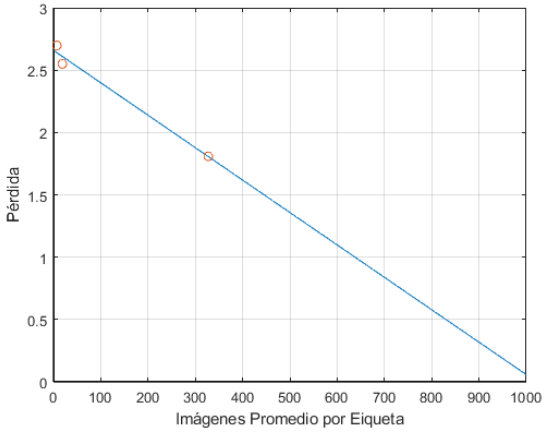
Exactitud. 5 Etiquetas



Pérdida. 5 Etiquetas



Exactitud. 15 Etiquetas



Pérdida. 15 Etiquetas

Figura 4.10: A través de las ecuaciones que grafican estas rectas (obtenidas de regresiones lineales basadas en los 3 puntos marcados por gráfica) se pudo predecir una cantidad de imágenes necesarias para una óptima predicción.

A través de las ecuaciones de las rectas obtenidas, es posible saber el número estimado de imágenes necesarias que se necesitarían para obtener los mejores resultados posibles, es decir, la cantidad de imágenes que pudieran dar la mayor exactitud o bien, la menor pérdida. Este estimado se aprecia en la Tabla 4.7.

Tabla 4.7: Estimado de Imágenes Requeridas

	De acuerdo a una óptima		
Número de Etiquetas Clasificatorias	Exactitud (100 %)	Pérdida (0.000)	Promedio de ambas
Para 3 Etiquetas	2207	2807	2507
Para 5 Etiquetas	2433	2716	2574
Para 15 Etiquetas	814	1022	918

En esta tabla se verifica lo que ya se sospechaba. Se requiere una mayor cantidad de imágenes si se trabaja con una mayor cantidad de etiquetas, pues a pesar de que la cantidad requerida de imágenes para cada etiqueta disminuye con más etiquetas, la suma total de imágenes es muy alta. Con 15 etiquetas y la cantidad promedio requerida para un óptimo, la cantidad estimada sería de: $15\ \texttt{etiquetas} \times \frac{918\ \texttt{imágenes promedio}}{\texttt{etiquetas}} = 13770$ imágenes para entrenar. El uso de una menor cantidad de etiquetas, junto con la propuesta del modelo doble serían una mejor opción. Considerado que el mismo conjunto de imágenes entrenaría a ambas etiquetas de flujo gaseoso y flujo líquido, entonces el mínimo necesario para 5 etiquetas sería suficiente también para 3 etiquetas. Por tanto la cantidad de 2574 imágenes en promedio por etiqueta para 5 clasificaciones resultan en 12870 totales. La diferencia parece mínima, pero se espera que el número baje drásticamente con la correcta configuración de metaparámetros de la CNN. Una cosa que es importante recalcar es que estas tendencias se hallaron mediante el uso de regresiones lineales, las cuales son muy sencillas como para englobar el comportamiento de un proceso tan complejo como una red neuronal.

4.7. Conclusión

Es muy importante hacer hincapié en la importancia de una buena cantidad de imágenes. Las estimaciones de cantidad de imágenes requeridas a través de regresiones lineales indican una cantidad de imágenes necesarias por etiqueta en bruto muy grandes. No se debe perder de vista que el estimado de imágenes utiliza un método muy sencillo y no alcanza a cubrir la complejidad de los procesos que realiza una CNN.

Otra cosa que es importante recalcar es que a través de la idea del modelo combinado, que utiliza 2 modelos secuenciales con 3 y 5 etiquetas, se obtuvieron mejores resultados que dividir la escasa cantidad de imágenes en 15 etiquetas. Se considera que la suma de nuevas imágenes a las que se les aplique el *"data augmentation"* junto con el modelo doble pueden llevar a que la CNN otorgue resultados bastante aceptables.

Capítulo 5

Propuesta del Sistema de Telemetría

En este capítulo se presenta una propuesta para ampliar el alcance de la CNN diseñada en esta tesis. Concretamente, el objetivo de este capítulo es describir un sistema de telemetría, que adquiera y envíe a una plataforma web con ayuda de una computadora `Raspberry Pi 3`, mediciones proporcionadas por sensores de presión. La idea es que estas mediciones sean utilizadas para generar imágenes de espectrogramas que sirvan de entrada de la CNN tal que ésta prediga los gastos másicos de un flujo bifásico. En la Figura 5.1 se ilustra el diagrama de los elementos que conforman este sistema de telemetría.

5.1. Transductores de Presión

Antes de presentar la propuesta, se discute brevemente sobre la naturaleza de los transductores o sensores de presión que se encuentran colocados en la tubería donde se tomaron los datos para entrenar la CNN propuesta en esta tesis.

Los sensores colocados a lo largo de la tubería son cuatro sensores de presión de la marca TE Connectivity serie U5300. Estos sensores son ideales para aplicaciones industriales y trabajos demandantes. Pueden medir presión de gas, líquido, o en nuestro caso, de flujos conformados por ambas fases líquidas y gaseosas. Posee exactitud del $\pm 0.1\,\%$, es a prueba

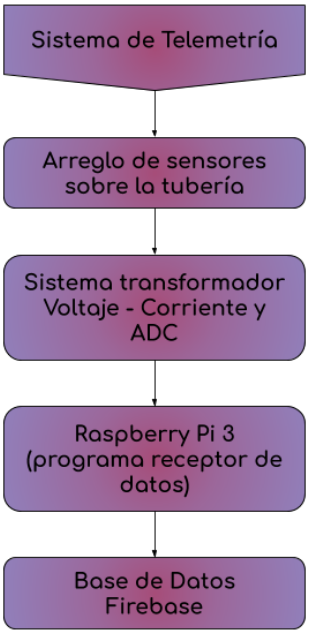


Figura 5.1: Sistema de Telemetría.

de agua, tiene protección contra cortos circuitos en la salida, y protección de polaridad inversa en su entrada. Su rango de lectura de presión está en el rango de 0 - 15 [PSI]. Su salida es una señal analógica entre 4 y 20 [mA]. Se alimenta con un voltaje de entre 9 - 30 [V]. Uno de los sensores se muestra en la Figura 5.2.

Figura 5.2: Transductor de presión U5300 (Hernández, 2019).

De esas características, la que es más relevante es la forma de la señal de salida: bucle de 4 a 20 [mA], que es el estándar principal en la industria en cuanto a señales analógicas.

5.2. Acoplamiento de los Transductores con la Tarjeta Raspberry Pi 3

Con la información que se encontró de los sensores de presión, sabemos que la salida de cada sensor funciona con bucle de corriente de 4 a 20 [mA]. Lo que se busca ahora es conectar la tarjeta `Raspberry Pi 3` a los cuatro sensores. Para ello debemos resolver dos asuntos: conectar cuatro sensores de salida por corriente a una computadora que solo acepta voltaje, y conectar esta salida analógica a un sistema que no posee un convertidor analógico digital (ADC).

5.2.1. Transformador Corriente - Voltaje

Para abordar el problema de la señal de corriente como entrada a un sistema que mide voltaje, debemos considerar la entrada que necesitamos. Buscaremos transformar la señal de corriente en una señal de voltaje en el rango de 1 a 5 [V]. Para ello se propone el uso del circuito transformador que se ilustra en la Figura 5.3.

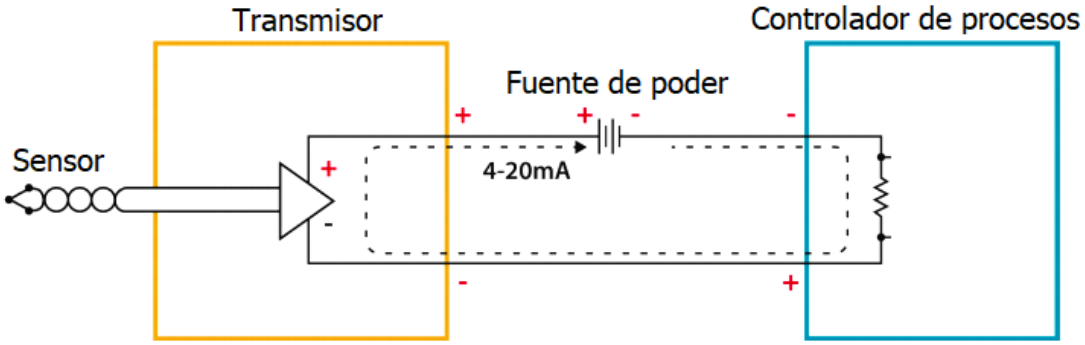


Figura 5.3: Circuito transformado Corriente → Voltaje (Vignesh, 2019).

El funcionamiento de este circuito es bastante simple, pues está basado en la Ley de Ohm. El circuito es alimentado con una fuente externa de voltaje de corriente directa (DC). El sensor actúa como el regulador de corriente del circuito, es decir, es quien dicta la corriente que circula por el circuito con base en las mediciones de presión. Cuando existe

una presión mínima, otorga 4 [mA] de corriente, y cuando mide la presión máxima otorga 20 [mA] de corriente. También actúa como una resistencia de descarga. La resistencia R_c donde se tomarán las medidas de voltaje que serán la entrada para la `Raspberry Pi 3` debe tener un valor de resistencia acorde a los valores de voltaje de salida que necesitamos, es decir, de 1 a 5 [V]. Entonces, para un voltaje de salida máximo $V_{out_{max}}$ se tiene que:

$$V = RI \rightarrow R_c = \frac{V}{I} = \frac{V_{out_{max}}}{I_{max}} = \frac{5[\text{V}]}{20[\text{mA}]} = 250[\Omega]. \tag{5.1}$$

Comprobamos con voltaje de salida mínima con corriente mínima:

$$V = RI \rightarrow R_c = \frac{V}{I} = \frac{V_{out_{min}}}{I_{min}} = \frac{1[\text{V}]}{4[\text{mA}]} = 250[\Omega]. \tag{5.2}$$

Nuestra resistencia de carga R_c debe ser una resistencia de valor 250[Ω]. Cuando el circuito está encendido, el sensor dicta la corriente, la fuente dicta la energía necesaria para que dicha corriente circule. El valor de la resistencia junto con la corriente dictan la caída de voltaje V_{out} que se lee en la salida, y la caída de voltaje restante la toma el sensor. De esta forma logramos traducir la señal analógica de corriente en una señal analógica de voltaje.

Dependiendo de las distancias de los cables involucrados y su resistencia inherente, puede que se tengan que hacer ajustes en los valores del cálculo. Una resistencia de protección de valores altos ($10[k\Omega]$, por ejemplo) es buena práctica a la salida. Este circuito se debe repetir cuatro veces, pues se tienen cuatro sensores.

Esta solución presenta la ventaja de ser simple y económica, considerando que el valor de cuatro resistencias depende de la calidad de las mismas. La desventaja de este método es la dificultad de la implementación, considerando la longitud de los cables y la posible posición de la `Raspberry Pi 3`, las resistencias y sobre todo la fuente de voltaje necesaria.

5.2.2. Convertidor Analógico Digital

Una vez que tenemos nuestra señal de voltaje, y no de corriente, se presenta un problema inherente de la computadora `Raspberry Pi 3`, y es que a diferencia de otros microcontroladores, computadoras o tarjetas, no posee un convertidor analógico digital (ADC, acrónimo del inglés *"analog to digital converter"*) que pueda leer una señal analógica y la interprete.

Una manera de abordar este problema es mediante el uso de una tarjeta ADS1115. Es un convertidor analógico digital de cuatro canales que trabaja con el protocolo I^2C y tiene una resolución de 15 bits. Junto a esta tarjeta, se usa un *"Level Shifter"* que simplemente modifica el voltaje de la señal de 5 [V] a 3.3 [V]. Esta última tarjeta no es cara y es un paso de seguridad para proteger la computadora.

Ambos, el ADS1115 y el *"level shifter"*, son tarjetas con cuatro canales. De tal forma que las conexiones y programación deben realizarse cuatro veces.

El ADS1115 se conecta a la `Raspberry Pi 3` a través de los puertos GPIO2 (SDA) y GPIO3 (SCL). Estos puertos se conectan al lado del voltaje bajo del *"level shifter"*, en los puertos A1 y A2. Del lado del voltaje alto, los puertos B1 y B2 se conectarán con los puertos SDA y SCL del ADS1115 como se observa en la Figura 5.4 (Rototron, 2016).

5.2.3. Protocolo I^2C

Es importante que se hable brevemente acerca del protocolo I^2C. Este protocolo, también denominado I2C o IIC, es un protocolo de comunicación serial. Como su nombre indica, permite el envío de información bit por bit de manera serial entre un dispositivo maestro (*"master"*) y un dispositivo esclavo (*"slave"*), a través de la línea SDA (*"serial data"*). Como otros protocolos, este es un sistema síncrono, es decir, la salida de información es sincronizada conforme al muestreo de los datos de la señal de un reloj entre maestro y esclavo, el cual se comparte mediante la línea SCL (*"serial clock"*). Son el o los dispositivos

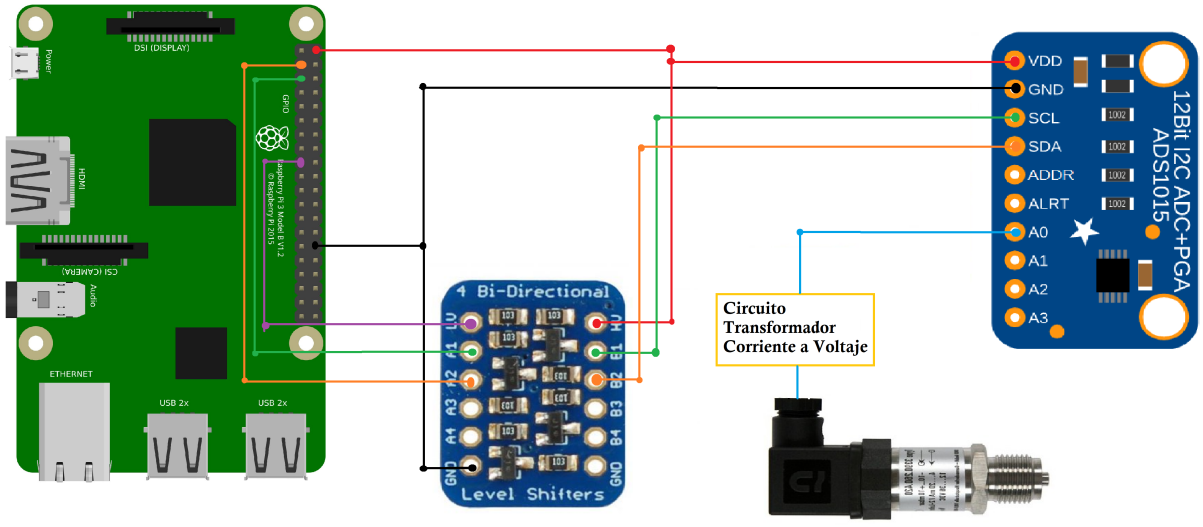


Figura 5.4: Diagrama de conexión propuesto para conectar la `Raspberry Pi 3` con el ADC a través del *"Level Shifter"*.

maestros los que tienen el control de la señal de reloj. La velocidad de transmisión entre dispositivos depende del modo de conexión. Distintos modos de conexión se encuentran en la Tabla 5.1.

Tabla 5.1: Modo de tasa de Transmisión

Modo	Tasa de transmisión
Modo Estándar	100 [kbps]
Modo Rápido	400 [kbps]
Modo de Alta Velocidad	3.4 [Mbps]
Modo de Velocidad Ultrarápida	5[Mbps]

Para iniciar el intercambio de información, el dispositivo maestro es quien inicia la comunicación siempre, y lo hace mediante una de dos secuencias especiales (siendo la otra la de terminación).

El protocolo I^2C transfiere la información en mensajes fraccionados en *"data frames"*. Cada mensaje tiene la dirección del dispositivo esclavo de la cual proviene y los *"data frames"* necesarios para enviar la información. La composición del mensaje I^2C se muestra en la Figura 5.5. En esta figura se pueden apreciar los bits de información en sintonía con los pulsos del reloj. De igual forma se puede apreciar las secuencias de inicio y término de la

transmisión. Se aprecia la estructura de la identificación, es decir, los 7 bits de la dirección, el bit de escribir/leer y el bit de reconocimiento (ACK). Seguido de estos se aprecian los bytes de información, cada uno seguidos por su respectivo bit de reconocimiento.

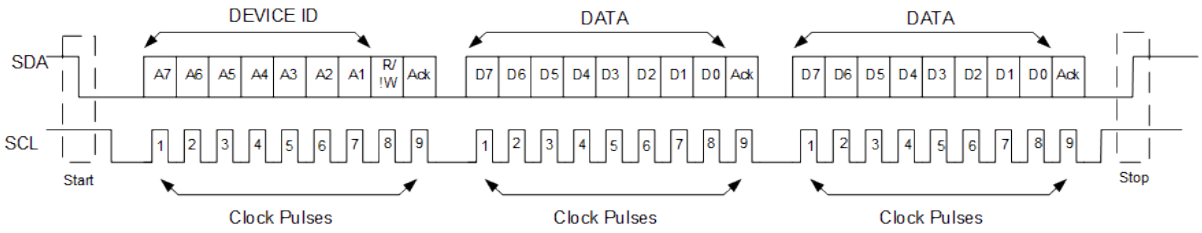


Figura 5.5: Estructura del datagrama I^2C (Lea, nd).

La dirección del dispositivo esclavo es de 7 o 10 bits. Es única para cada dispositivo. El bit de escritura/lectura identifica si el maestro está enviando datos o solicitándolos. Los bits de ACK/NACK son banderas que permiten saber si el paquete se ha recibido con éxito o no.

5.2.4. Módulo Alternativo

Se propone una segunda alternativa a la propuesta anterior. La nueva propuesta sería la adquisición de un módulo para la `Raspberry Pi 3` que se encargue directamente del bucle de corriente y de la transición analógica a digital, y sólo haga falta conectar los sensores directamente a este módulo. Esta opción tiene la desventaja del coste del módulo pero, la ventaja de la facilidad de implementación.

En cuanto a su funcionamiento, funciona como todo lo anteriormente mencionado, pero ya hecho para el comprador, implementado en un cómodo módulo listo para ser instalado.

En la Figura 5.6 se ilustra uno de estos módulos del fabricante NCD (*"National Control Devices"*). Existen más módulos de este vendedor, sin embargo este se acoplaría perfecto por tener cuatro entradas para corriente y bus I^2C, entre otras características.

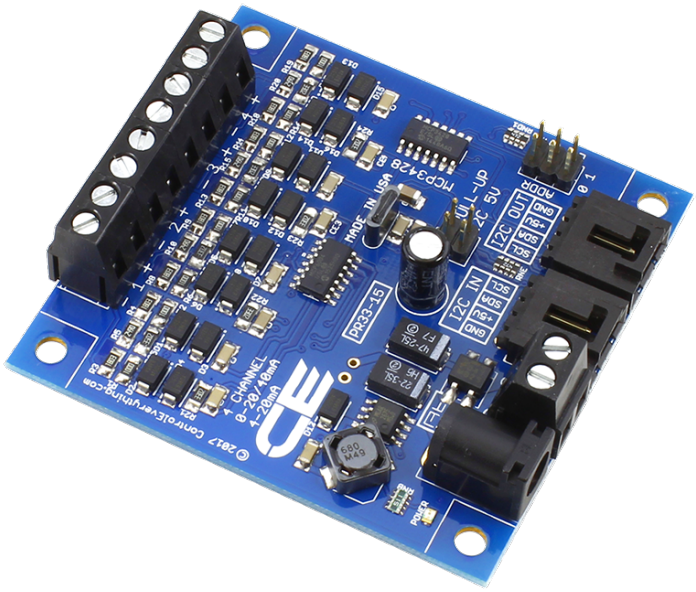


Figura 5.6: Modulo *4-Channel I2C 4-20mA Current Receiver with I2C Interface* para la `Raspberry Pi 3` . Este módulo se encarga de la recepción de datos. Posee un ADC y puede recibir señales de corriente (NCD-Store, 2020).

5.3. Recepción y Envío de Datos

Una vez que el sistema se ha implementado en términos de hardware, debemos pasar al software. El problema es recibir los datos que se están obteniendo de los sensores para almacenarlos o dirigirlos a un lugar en internet de donde se puedan descargar posteriormente. El lugar en el internet elegido es la Plataforma `Firebase`. `Firebase` es un servicio de Google el cual maneja bases de datos y gestiona información en tiempo real. Se escogió esta plataforma de entre otras por su actualización en tiempo real de sus bases de datos, a pesar de su costo.

5.3.1. Propuesta de Código para la Obtención de Datos y su Almacenamiento en la Plataforma `Firebase`

El objetivo es indicarle a la `Raspberry Pi 3` que en sus puertos SDA y SCL se acercan datos obtenidos de sensores. Para ello trabajamos con un programa en Python 3. Se puede

realizar en cualquier ambiente de programación de Python, pero se debe ejecutar en la terminal de la `Raspberry Pi 3`. El código se observa en el Anexo B, pero es importante hablar de su propósito. En términos sencillos, el programa es un ciclo que funciona cuando se requiere. La computadora toma el valor que se halle almacenado en el ADC, abre una conexión con la plataforma `Firebase`, quien toma el dato cada que la información se actualiza, lo almacena y cierra la conexión. En tanto el ciclo esté encendido (es decir hasta que se detenga la ejecución del programa) la plataforma se actualizará en tiempo real.

Se importan tres bibliotecas:

1. Biblioteca `Firebase`. Esta biblioteca contiene las funciones para conexión con la plataforma.
2. `Sleep` desde la biblioteca `Time`. La función `sleep` permite dar una pausa a la iteración del ciclo.
3. Biblioteca `Adafruit_ADS1x15`. Esta biblioteca permite la conexión entre la computadora con el ADS1115.

En la sección anterior, se discutió un poco acerca del funcionamiento del protocolo I^2C. La biblioteca `Adafruit_ADS1x15` es la biblioteca que se conecta con el bus serial I^2C. Utilizamos una función de esta biblioteca, para mandar a llamar el bus, a través de una dirección: la dirección $0x48$. Con esta sencilla línea, estamos en contacto con el ADS1115.

La siguiente línea de código es referente a la comunicación con la plataforma `Firebase`. Solo se están declarando la variable que contiene el nombre de la página y nuestra clave personal del canal a donde se transmitirán los datos.

La siguiente sección del código es el bucle principal.

Al comienzo indicamos el camino para 4 canales, pues los 4 canales del ADS serán la fuente de la información. En este caso, para 4 sensores, los datos vienen desde el canal 0, 1, 2 y 3. A través del función `adc.read`, tomamos el valor que se encuentre en el ADC en ese momento.

Ahora debemos preguntarnos, qué forma tiene este valor. Como se mencionó anteriormente, el ADC es el modelo 1115. Este modelo tiene 15 bits de resolución y su bit de signo. Es decir, este convertidor analógico digital toma los rangos físicos del valor de voltaje de salida V_{out} y los traduce en toda su resolución. El valor que la variable toma es un valor decimal entre -32768 y 32768 ($\pm 2^{15}$). Para volver a obtener el valor de voltaje, debemos normalizar este valor, es decir dividirlo en 32768. Por cuestiones de ganancia, ese valor debe multiplicarse por 6.144. Con ello, tenemos el valor de voltaje necesario.

La siguiente línea del código es referente a la transformación entre la variable de salida en valores de presión, para ello utilizamos nuestros nuevos valores de voltaje y calcularemos la regla de correspondencia entre voltaje y presión.

Con base en los valores máximos y mínimos del voltaje de salida, V_{out}, (corriente transformada en voltaje) así como los valores de presión máxima y mínima que el sensor puede medir, podemos trazar la regla de correspondencia. El rango de voltaje de nuestra medición es de 1 a 5 [V]. La presión máxima que el sensor puede medir es de 15 [PSI] y la mínima 0. Conociendo los 2 puntos de la gráfica, podemos encontrar la regla de correspondencia.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{15 - 0}{5 - 1} = 3.75 \tag{5.3}$$

$$y = mx + b \rightarrow b = y - mx = 15 - 3.75(5) = -3.75 \tag{5.4}$$

$$P = 3.75v - 3.75, \tag{5.5}$$

Donde P es la presión de salida medida en [psi] y v es el voltaje de salida V_{out} como la variable independiente de la recta.

Como se mencionó, la plataforma que elegimos para enviar la información de las presiones es `Firebase`. Para el envío de datos se utiliza una biblioteca especializada que permite la apertura de un canal de comunicación entre la plataforma y la `Raspberry Pi`, resultando que la trasmisión de datos se vuelva muy sencilla. La biblioteca en cuestión es la antes mencionada `Firebase`. Este módulo define clases y funciones necesarias para la correcta

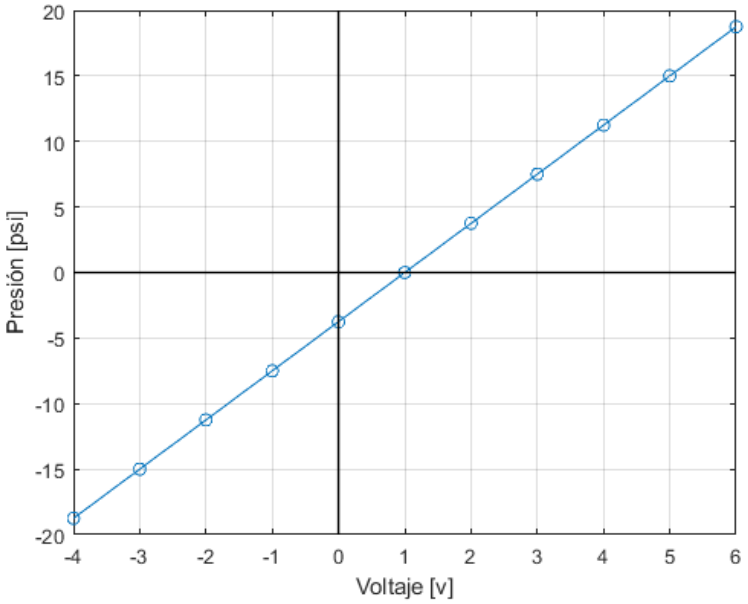


Figura 5.7: Gráfica del comportamiento del sensor.

apertura del canal de comunicación entre la `Raspberry Pi 3` y la plataforma.

Cada vez que existe un cambio en la variable en el *"loop"* (bucle), se envía un dato a la plataforma a través de un *"websocket"*. Un *"websocket"* es un protocolo de comunicación TCP (es decir, orientado a conexión, o bien, se encarga de que la información llegue y de no hacerlo solicita reenvío). Es un protocolo que tiene un *"handshake"* (establecimiento de canal de comunicación) semejante a HTTP. Lo que hace es abrir un canal de comunicación *"full duplex"*. Al ser bidireccional, la información se envía inmediatamente se ha establecido la conexión (Simões, 2019). Los *"websocket"* son más veloces que una solicitud HTTP (Ashwini, 2017). El dato que se envía toma la forma de un archivo tipo JSON.

Un archivo de tipo JSON (de *"JavaScript Object Notation"*) es un tipo de archivo especializado en el envío y recepción de datos. Este archivo usa texto plano para almacenar y enviar datos, los cuales están constituidos por atributos y valores o vectores de datos.

Finalmente, el programa cierra la conexión, y pone una pausa. Luego, el ciclo continua, tomando valores del bus serial cada que el ciclo corre. Este ciclo se mantiene mientras el programa esté ejecutando. Mientras tanto en la plataforma, lentamente van apareciendo

los valores que le estamos mandando. La lista de valores de presión que se van obteniendo se pueden descargar.

No debemos olvidar que esta fase del proyecto es una propuesta. La enorme ventaja de esta propuesta es la velocidad con que los datos se suben a la plataforma. La base de datos de `Firebase` se actualiza en milisegundos, es decir, en tiempo real. La desventaja de esta plataforma es su precio. Es una plataforma basta y robusta, pero para los fines de la CNN, se encontró el posible precio (puede variar, basado en el verdadero uso de la propuesta) el cual es de USD 5 por cada GB de información que se envíe (Google, 2020).

Capítulo 6

Conclusiones

En esta tesis se demostró que entrenar una CNN para predecir (identificar) los gastos másicos de un flujo bifásico (glicerina y aire), a partir de imágenes de espectrogramas de diferenciales de presión, es totalmente factible.

El factor determinante para el correcto entrenamiento de una CNN es la cantidad de imágenes, por lo que es crucial contar con un gran conjunto de imágenes para alcanzar una buena predicción. En esta tesis, a pesar de la falta de imágenes, la cantidad utilizada para el entrenamiento fue suficiente para generar el conocimiento necesario para la identificación de los gastos másicos. Para lograrlo, y dado que no existe una fórmula que identifique la cantidad precisa de imágenes necesarias para un entrenamiento exitoso, se buscó un estimado óptimo de imágenes necesarias para el entrenamiento. La estimación se basó en el éxito (exactitud y pérdida de prueba) que tuvo la red con respecto a la cantidad de etiquetas.

Concretamente, el número estimado de imágenes promedio por etiqueta necesarias para obtener resultados aceptables se realizó con base a las tendencias obtenidas a través de simples regresiones lineales. Se demostró que el número de imágenes requerido para un entrenamiento exitoso aumenta si el número de etiquetas aumenta también.

Los dos factores que fueron fundamentales para el entrenamiento exitoso de la CNN

propuesta en esta tesis fueron el *"data augmentation"* y la asignación de la tarea de predicción a un modelo doble. El *"data augmentation"* es un proceso exógeno en el que se modifican (filtran) las imágenes originales para aumentar el tamaño del conjunto de imágenes a usar en el entrenamiento. Este proceso no lo hace la CNN, sino se hace de manera independiente antes de comenzar el entrenamiento.

Acerca del sistema de telemetría, la propuesta se vislumbra funcional, pero se requiere implementarlo en laboratorio para saber si es una propuesta rentable. Se buscó que la propuesta fuera económica, pero hasta no implementarla, no es posible estimar su valor útil.

La propuesta a futuro es que este sistema pueda enviar los datos de presión en tiempo real para que se procesen como imágenes que sirvan para estimar los gasto másicos.

Resumen del Artículo "*Evolution of High-Viscosity Gas–Liquid Flows as Viewed Through a Detrended Fluctuation Characterization*"

En este artículo, los autores Jonathan Hernández, Diego Galaviz, Lizeth Torres, Arturo Palacio Pérez, Alejandro Rodríguez Valdés y José Enrique Guzmán, propusieron caracterizar flujos intermitentes líquido-gas de alta viscosidad utilizando el análisis de fluctuaciones sin tendencia (DFA, acrónimo de *"Detrended Fluctuation Analysis"*). En particular, se caracterizó el flujo de una mezcla conformada de glicerina (con una viscosidad dinámica $\eta = 1.1\text{Pa} \cdot \text{s}$ a una temperatura de 25°C) y aire.

Para ello, los autores utilizaron medidas de presión adquiridas en diferentes coordenadas a lo largo de una tubería experimental del Instituto de Ingeniería. Luego, se aplicó el DFA a las series de tiempo de presión para caracterizar los patrones de flujo asociados a la mezcla bifásica. El resultado de la caracterización fue un conjunto de leyes de potencia cuyos exponentes se asociaron a un patrón de flujo.

Aparato de Prueba

Los experimentos se llevaron a cabo en el bucle de flujo que se aprecia en la Figura 1. En este bucle, la fase líquida se suministra a la sección de prueba a través de una bomba de cavidad progresiva (Seepex Mod. BN35-24). Esta bomba es capaz de entregar gastos másicos constantes en el intervalo de $0.0[kg/s]$ a $6.1[kg/s]$. La salida de la sección de prueba se conecta a un tanque separado con capacidad interna de $1.5[m^3]$. Por otro lado, un compresor Kaeser Aircenter SK.2 suministra una masa constante de aire seco a temperatura ambiente a una presión en el intervalo de $0.0[Pa]$ a $1.6 \times 10^6[Pa]$. El gasto másico se configura cuidadosamente con un regulador y válvulas de globo. La mezcla se produce en la conexión de 3 vías mostrado en la Figura 1. Los gastos másicos se pueden medir a la entrada con un caudalímetro de coriolis Endress-Hauser. Todas las presiones se miden con un arreglo de transductores convencionales marca MEAS U5300. Los intervalos de medición seleccionados para estos instrumentos son de $0.0[Pa]$ a $1.03 \times 10^5[Pa]$ y de $0.0[Pa]$ a $3.45 \times 10^5[Pa]$.

Metodología

Los experimentos se produjeron de acuerdo a la matriz de gastos másicos indicados en la Tabla 1. En total, 15 diferentes combinaciones (q_s, q_l) se consideraron.

Tabla 1: Matriz de experimentos. Gastos másicos líquido (q_l) y gaseoso (q_g) insertadas en la sección de prueba.

$q_g[kg/s]$	$q_l[kg/s]$
	1.3
0.005	2.5
0.01	3.7
0.015	4.9
	6.1

A pesar de que el flujo bifásico se desarrolló rápidamente, al sistema se le permitió reposar por algunos minutos antes de tomar cualquier tipo de medida. A este punto las

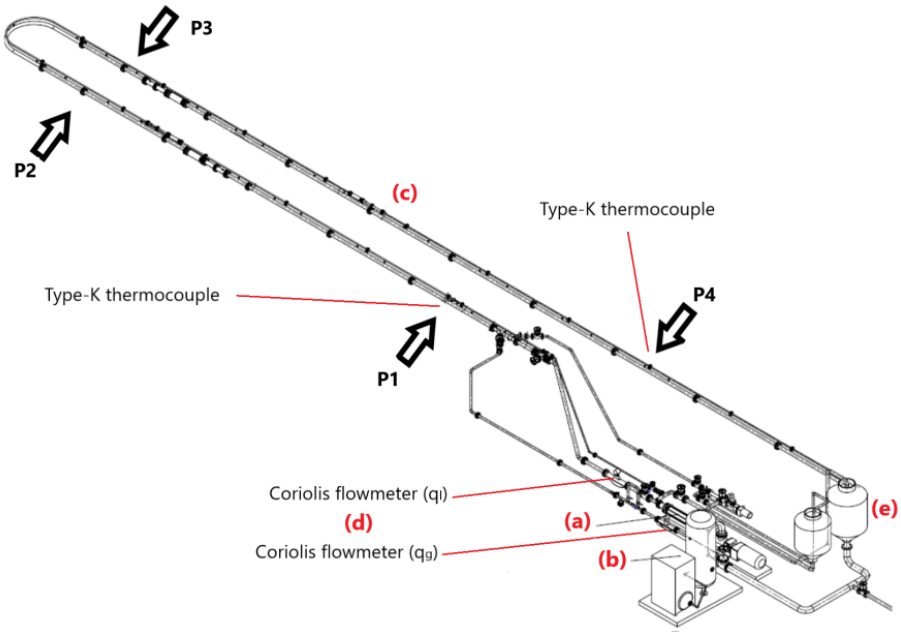


Figura 1: Aparato de pruebas. Los elementos principales del bucle de flujo son: a) subsistema de bombeo, b) subsistema de gas presurizado, c) bucle de flujo, d) entrada de flujo del sistema de medición, y e) separador y tanques de almacenaje. El bucle de flujo consiste en una tubería de 54 m de largo, junto con los transductores de presión, termocoples e instrumentación para tomografías.

propiedades del flujo se midieron y colectaron con el sistema de adquisición de datos.

Para remover cualquier especie de influencia externa que pudiera afectar las medidas de presión, solo se tomaron presiones diferenciales, las cuales fueron usadas para determinar los gradientes de presión de interés. Los puertos de medición se localizaron a 0, 18, 22 y 43 $[m]$ aguas abajo de la entrada. Estos puertos se etiquetaron como P1, P2, P3 y P4 respectivamente en la Figura 1.

Alrededor de 900 muestras se recolectaron para las variables medidas por experimento. El conjunto universal contuvo cerca de 750,000 datos, consistiendo de medidas simultáneas de flujo másico presiones y temperaturas.

Información Utilizada en esta Tesis

Los datos obtenidos en este artículo fueron utilizados en esta tesis para obtener los espectrogramas. Los conjuntos de imágenes mostrados en las siguientes tres figuras muestran los espectrogramas de las presiones correspondientes a los gastos másicos de entrada de la matriz experimental (Tabla 1). Es importante notar que los tres conjuntos corresponden a las tres secciones delimitadas por los sensores de presión instalados a lo largo del tubo. Las columnas representan q_l y las filas q_g.

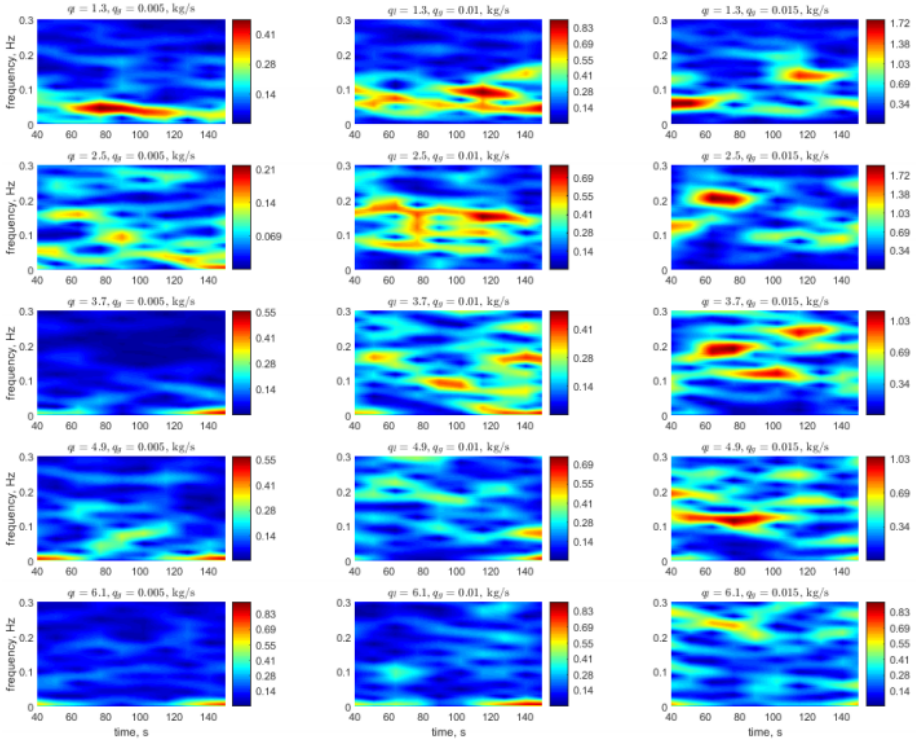


Figura 2: Matriz de espectrogramas para $\Delta P_{12} = P1 - P2$. El rango de la barra de color indica el nivel de presión en $[kPa]$. Es importante notar que para propósitos de visualización los rangos respectivos no están normalizados. Estos espectrogramas indican como diferentes bandas de frecuencia se excitan a diferentes tiempos en una locación específica, mientras que todo el conjunto de imágenes ilustra la diferencia de una locación a otra.

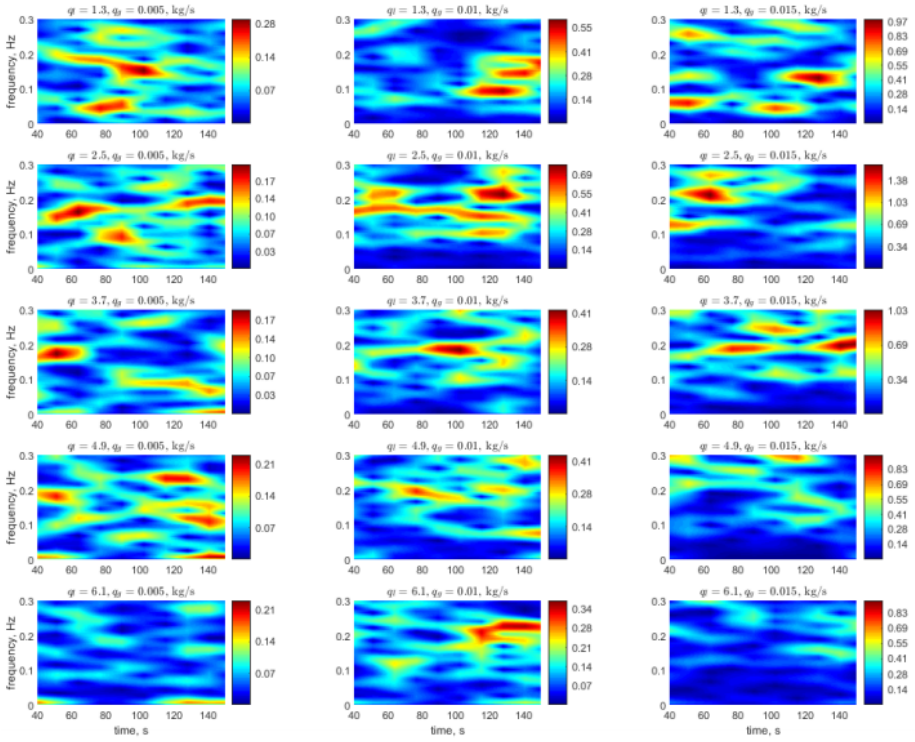


Figura 3: Matriz de espectrogramas para $\Delta P_{23} = P2 - P3$. El rango de la barra de color indica el nivel de presión en $[kPa]$. Es importante notar que para propósitos de visualización los rangos respectivos no están normalizados. Estos espectrogramas indican como diferentes bandas de frecuencia se excitan a diferentes tiempos en una locación específica, mientras que todo el conjunto de imágenes ilustra la diferencia de una locación a otra.

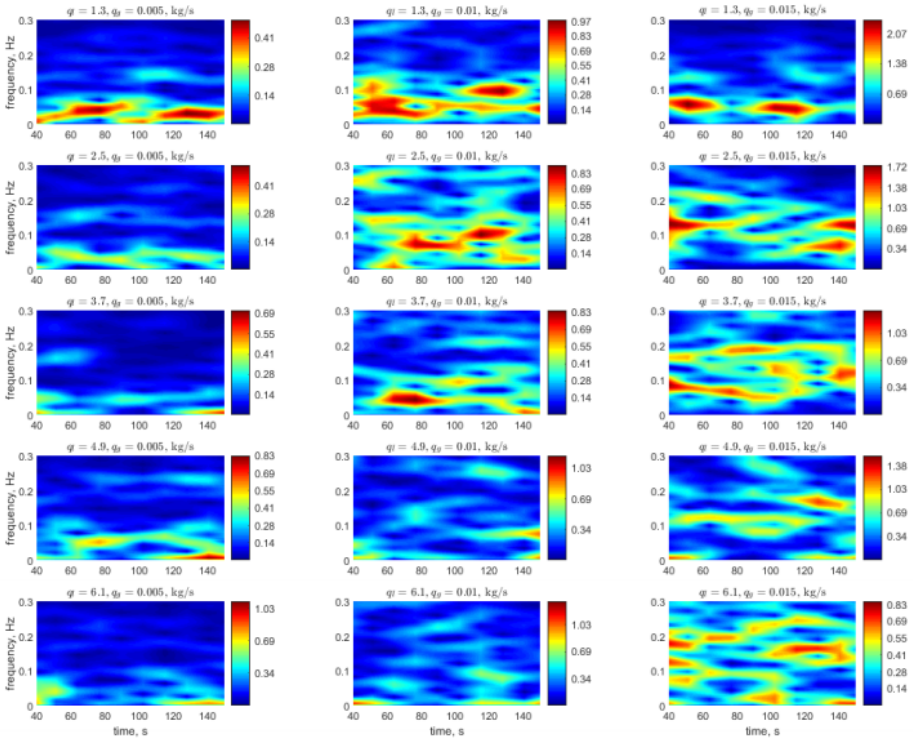


Figura 4: Matriz de espectrogramas para $\Delta P_{34} = P3 - P4$. El rango de la barra de color indica el nivel de presión en $[kPa]$. Es importante notar que para propósitos de visualización los rangos respectivos no están normalizados. Estos espectrogramas indican como diferentes bandas de frecuencia se excitan a diferentes tiempos en una locación específica, mientras que todo el conjunto de imágenes ilustra la diferencia de una locación a otra.

B. Códigos

Código General de la Red Neuronal Convolucional

In [3]:
```
import tensorflow as tf
import numpy
import os
import re
import keras
from keras.utils import to_categorical
from keras.models import Sequential,Input,Model
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.layers.normalization import BatchNormalization
from keras.models import load_model as lm
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from skimage.transform import resize
```

In [3]:
```
dirname = os.path.join(os.getcwd(), '306_15etiquetas') #indicar carpeta en la que están las imágenes
imgpath = dirname + os.sep

images = []
directories = []
dCount = []
count=0
PreviousRoot=''

print("Cargando imagenes de",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg)$", filename):
            count=count+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            image_resized = resize(image, (56, 56),preserve_range=True)
            images.append(image_resized)

            if PreviousRoot !=root:
                PreviousRoot=root
                directories.append(root)
                dCount.append(count)
                count=0
dCount.append(count)

dCount = dCount[1:]
dCount[0]=dCount[0]+1
print("Imagenes en cada directorio", dCount)
print('Total de imágenes:',sum(dCount))
```

Figura 5: Código de la Red Neuronal Convolucional. Parte 1.

In [13]:
```
labels=[]
index=0
for amount in dCount:
    for i in range(amount):
        labels.append(index)
    index=index+1
print("Cantidad de etiquetas creadas: ",len(labels))

gr=[]
index=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(index , name[len(name)-1])
    gr.append(name[len(name)-1])
    index=index+1

X = numpy.array(images, dtype=numpy.uint8) #convertir las imágenes a array
Y = numpy.array(labels)

# Asignar las clases a cada carpeta
classes = numpy.unique(Y)
nClasses = len(classes)
print('Clases de salida: ', classes)
```

In [15]:
```
#Se dividen las imágenes en entrenamiento y prueba
train_image,test_image,train_label,test_label = train_test_split(X,Y,test_size=0.15,random_state=0)

train_image = train_image.astype('float32')
test_image = test_image.astype('float32')
train_image = train_image / 255. #Normalizar el arreglo
test_image = test_image / 255.

# Cambiar las etiquetas con one-hot encoding
train_label_OH = to_categorical(train_label)
test_label_OH = to_categorical(test_label)

# Visualizar ejemplo del nombre original de la etiqueta y su nombre one-hot
print('Etiqueta original:', train_label[0])
print('One-hot:', train_label_OH[0])

#Se dividen las imágenes en entrenamiento y validación
train_image,valid_image,train_label,valid_label = train_test_split(train_image, train_label_OH, test_size=0.15, random_state=0)
```

In [16]:
```
a_model = Sequential() #Se usa el modelo secuencial
a_model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding='same',strides=(2,2),input_shape=(56,56,3)))#Convoluciones
a_model.add(MaxPooling2D((3, 3),padding='same'))#se reduce el tamaño de la proxima capa
a_model.add(Dropout(0.25)) #elimina un porcentaje de neuronas
a_model.add(BatchNormalization())

a_model.add(Flatten()) #conector entre capa convolucional y tradicional
a_model.add(Dense(32, activation='softmax')) #capa tradicional
a_model.add(Dropout(0.3))
a_model.add(Dense(15, activation='softmax'))#se asignan probabilidades

a_model.summary() #imprime la tablita

a_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(lr=0.0002), metrics=['accuracy'])
```

In [17]:
```
a_train_dropout = a_model.fit(train_image, train_label, epochs=100, verbose=1, validation_data=(valid_image, valid_label))

# guardamos la red, para utilizarla en el futuro, sin tener que volver a entrenar
a_model.save("prueba.h5")
```

In [13]:
```
test = a_model.evaluate(test_image, test_label_OH)

print('Pérdida:', test[0])
print('Exactitud:', test[1])
```

Figura 6: Código de la Red Neuronal Convolucional. Parte 2.

Código para la Predicción del Modelo Doble

In [1]:
```
import tensorflow as tf
import numpy
import os
import re
import keras
import matplotlib.pyplot as plt
%matplotlib inline
from skimage.transform import resize
from keras.models import load_model
from keras.preprocessing import image
```

In [13]:
```
def load_image(img_path, show=False):

    img = image.load_img(img_path, target_size=(56, 56))#Asignar tamaño de la imagen
    img_tensor = image.img_to_array(img)                 #Convertir imagen a array
    img_tensor = numpy.expand_dims(img_tensor, axis=0)  #Añadir eje
    img_tensor /= 255.              #Normalizar

    if show:
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()

    return img_tensor

# cargar modelo
a_model = load_model("aire.h5")

# ruta de la imagen
img_path = '2331.jpg'

# cargar la imagen
new_image = load_image(img_path)

classes = a_model.predict_classes(new_image) #predecir clase

print (classes) #imprimir predicción
```

In [14]:
```
def load_image(img_path, show=False):

    img = image.load_img(img_path, target_size=(56, 56)) #Asignar tamaño de la imagen
    img_tensor = image.img_to_array(img)                 #Convertir imagen a array
    img_tensor = numpy.expand_dims(img_tensor, axis=0)  #Añadir eje
    img_tensor /= 255.               #Normalizar

    if show:
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()

    return img_tensor



# cargar modelo
g_model = load_model("glicerina.h5")

# ruta de la imagen
img_path = '2331.jpg'

# cargar la imagen
new_image = load_image(img_path)

classes2 = g_model.predict_classes(new_image) #predecir clase

print (classes2) #imprimir predicción
```

Figura 7: Código de la Red Neuronal Convolucional en su forma de Modelo Doble.

Código para la Propuesta de Telemetría

```
from time import sleep
from firebase import firebase
import Adafruit_ADS1x15
adc = Adafruit_ADS1x15.ADS1115(address=0x48, busnum=1)
GAIN = 2/3

#colocar url del proyecto en firebase
firebase = firebase.FirebaseApplication('https://YOUR_FIREBASE_URL.firebaseio.com/', None)

while True:
    try:
        value = [0]*4

        value[0] = adc.read_adc(0, gain=GAIN)
        # Se determina el voltaje máximo
        volts = value[0] / 32767.0 * 6.144
        # Relación lineal entre presión y voltaje
        presion1 = 3.75 * volts - 3.75

        value[1] = adc.read_adc(1, gain=GAIN)
        volts = value[1] / 32767.0 * 6.144
        presion2 = 3.75 * volts - 3.75

        value[2] = adc.read_adc(2, gain=GAIN)
        volts = value[2] / 32767.0 * 6.144
        presion3 = 3.75 * volts - 3.75

        value[3] = adc.read_adc(3, gain=GAIN)
        volts = value[3] / 32767.0 * 6.144
        presion4 = 3.75 * volts - 3.75

        # Si la lectura es válida
        if presion1 is not None and presion2 is not None and presion3 is not None and presion4 is not None:

            data = {"Presion1": presion1, "Presion2": presion2, "Presion3": presion3, "Presion4": presion4}
            #su suben los datos a firebase
            firebase.put('/sensor', data)
        else:
            print 'Error'

        sleep(0.2)
    except:
        break
```

Figura 8: Código de la Propuesta de Red de Telemetría.

Referencias

Prabhu, Understanding of Convolutional Neural Network (CNN), https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148, 2018.

S. Sharma, Activation functions in neural networks, International Journal of Engineering Applied Sciences and Technology 6.

A. Amidi, S. Amidi, CS 229 - Machine Learning Tips and Tricks Cheatsheet, https://stanford.edu/ shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricksclassification-metrics, 2018.

J. Hernández, Estudio del Mecanismo de Transporte Predominante en Oleoductos Horizontales de Crudo Pesado con Presencia de Inhibidores de Arrastre, Ph.D. thesis, Universidad Nacional Autónoma de México, 2019.

Vignesh, 4-20 mA Current Loop - Distributed Control Systems - DCS - Instrumentation Forum, https://instrumentationforum.com/t/4-20-ma-current-loop/8569, 2019.

Learn.Digilentinc | Project 8: chipKIT^{TM} Pro and Serial Communications, https://learn.digilentinc.com/Documents/199, n.d.

NCD-Store, 4-Channel I2C 4-20mA Current Receiver with I2C Interface - store.ncd.io, https://store.ncd.io/product/4-channel-i2c-4-20ma-current-receiver-with-i2c-interface/, 2020.

J. Hernández, D. Galaviz, L. Torres, A. Palacio-Pérez, A. Rodríguez-Valdés, J. Guzmán, Evolution of High-Viscosity Gas–Liquid Flows as Viewed Through a Detrended Fluctuation Characterization, Processes 7 (11) (2019) 822.

R. D. Caughron, Two-phase pressure drop: a literature survey and correlation analysis, Master's thesis, Kansas State University, 1967.

G. E. Alves, Cocurrent liquid-gas flow in a pipe-line contactor, Chemical Engineering Progress 50 (9) (1954) 449–456.

G. Matsui, Identification of flow regimes in vertical gas-liquid two-phase flow using differential pressure fluctuations, International journal of multiphase flow 10 (6) (1984) 711–719.

M. Du, H. Yin, X. Chen, X. Wang, Oil-in-water two-phase flow pattern identification from experimental snapshots using convolutional neural network, IEEE Access 7 (2018) 6219–6225.

Z. Xu, F. Wu, X. Yang, Y. Li, Measurement of Gas-Oil Two-Phase Flow Patterns by Using CNN Algorithm Based on Dual ECT Sensors with Venturi Tube, Sensors 20 (4) (2020) 1200.

S. Torisaki, S. Miwa, Robust bubble feature extraction in gas-liquid two-phase flow using object detection technique, Journal of Nuclear Science and Technology (2020) 1–14.

K. Fukushima, S. Miyake, Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, in: Competition and cooperation in neural nets, Springer, 267–285, 1982.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

W. Burger, M. J. Burge, Digital Image Processing. An Algorithmic Introduction Using Java, Springer, 2016.

A. Behl, A. Bhatia, A. Puri, Convolution and Applications of Convolution, International Journal of Innovative Research in Teclmology 1 (6).

I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

J. Torres, DEEP LEARNING Introducción práctica con Keras. Primera parte., Lulu.com, 2018.

X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, 249–256, 2010.

P. Skalski, Gentle Dive into Math Behind Convolutional Neural Networks, https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9, 2019.

J. Brownlee, A Gentle Introduction to the Rectified Linear Unit (ReLU), https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, 2019.

M. Basavarajaiah, Maxpooling vs minpooling vs average pooling, https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9, 2019.

S. Nayak, Batch Normalization in Deep Networks | Learn OpenCV, https://www.learnopencv.com/batch-normalization-in-deep-networks/, 2018.

S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 .

C. Versloot, How does the Softmax activation function work? – MachineCurve, https://www.machinecurve.com/index.php/2020/01/08/how-does-the-softmax-activation-function-work/, 2020.

V. Kulkarni, Cross-Entropy for Dummies. A simple and intuitive explanation of information, entropy, and cross-entropy for data scientists, https://towardsdatascience.com/cross-entropy-for-dummies-5189303c7735, 2019.

DaneriCoding, ML.04. Red Neuronal Básica | Redes Neuronales, https://danerineuronal.wordpress.com/2017/09/16/ml-04-red-neuronal-basica/, 2017.

X. Wang, E. Kodirov, Y. Hua, N. Robertson, Instance Cross Entropy for Deep Metric Learning, arXiv preprint arXiv:1911.09976 .

F. Chollet, et al., SGD, https://keras.io/api/optimizers/sgd/, 2015a.

D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 .

F. Chollet, et al., Adam, https://keras.io/api/optimizers/adam/, 2015b.

J. Torres, DEEP LEARNING Introducción práctica con Keras. Segunda parte, Lulu.com, 2019.

J. I. Bagnato, Clasificación de Imágenes en Python | Aprende Machine Learning, https://www.aprendemachinelearning.com/clasificacion-de-imagenes-en-python/, 2018.

Rototron, Raspberry Pi Analog Water Sensor Tutorial, https://www.rototron.info/raspberry-pi-analog-water-sensor-tutorial/, 2016.

C. Simões, REST vs WebSocket. ¿Qué diferencias hay?, https://www.itdo.com/blog/rest-vs-websocket-que-diferencia-hay/, 2019.

A. Ashwini, Using Firebase To Provide Real-time Notifications, https://medium.com/swlh/using-firebase-to-provide-real-time-notifications-2f86b2f7d499, 2017.

Google, Understand Realtime Database Billing Firebase Realtime Database, https://firebase.google.com/docs/database/usage/billing, 2020.

Bibliografía

M. R. Islam, M. H. Chaudhry, Modeling of constituent transport in unsteady flows in pipe networks, Journal of Hydraulic Engineering 124 (11) (1998) 1115–1124.

E. Sletfjerding, J. S. Gudmundsson, Friction factor directly from roughness measurements, Journal of energy resources technology 125 (2) (2003) 126–130.

X. Xu, B. Karney, An Overview of Transient Fault Detection Techniques, in: Modeling and Monitoring of Pipelines and Networks, Springer, 13–37, 2017.

G. O. Brown, The history of the Darcy-Weisbach equation for pipe flow resistance, in: Environmental and Water Resources History, 34–43, 2003.

C. M. H., Applied Hydraulic Transients, Springer, third edition edn., ISBN 978-1-4614-8537-7, 2014.

B. E. Larock, R. W. Jeppson, G. Z. Watters, Hydraulics of pipeline systems, CRC press, 1999.

B. Kingdom, R. Liemberger, P. Marin, The challenge of reducing non-revenue water (NRW) in developing countries, 2006.

R. Lopezlena, Computer implementation of a boundary feedback leak detector and estimator for pipelines I: Transient Model, in: Mems. 16o Congreso Latinoamericano de Control Automático (CLCA 2014), 14–17, 2014.

Basics of the I2C Communication Protocol., https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/, n.d.

Using the I2C Bus, https://www.robot-electronics.co.uk/i2c-tutorial, n.d.

S. Paonessa, B. McDuffee, Back to Basics: The Fundamentals of 4-20 mA Current Loops, https://www.predig.com/indicatorpage/back-basics-fundamentals-4-20-ma- current-loops, 2020.

R. Lockhart, How To Make 4-20 mA Current Loop Measurements, https://www.dataq.com/blog/data-acquisition/4-20-ma-current-loop- measurements/, 2013.

A. Barón, Monitorización de la energía consumida mediante Raspberry Pi para sistema domótico, bachelor Thesis, 2017.

Google, Curso intensivo de aprendizaje automático | Google Developers, https://developers.google.com/machine-learning/crash-course/ glossary?hl=es-419, 2020.

V. Powell, Image Kernels Explained Visually, 2016.

R. Karim, 10 Stochastic Gradient Descent Optimisation Algorithms + Cheat Sheet, https://towardsdatascience.com/10-gradient-descent-optimisation- algorithms-86989510b5e9, 2018.

D. Shulga, Exploring Activation Functions for Neural Networks, https://towardsdatascience.com/exploring-activation-functions-for-neural -networks-73498da59b02, 2017.

R. Ruizendaal, Deep Learning #3: More on CNNs & Handling Overfitting, https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling -overfitting-2bd5d99abe5d, 2017.

F. D., Batch normalization in Neural Networks, https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c, 2017.

J. I. Bagnato, Convolutional Neural Networks: La Teoría explicada en Español | Aprende Machine Learning, https://www.aprendemachinelearning.com/como-funcionan-las-convolutional -neural-networks-vision-por-ordenador/, 2018.

J. Martínez, ¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning? - DataSmarts Español, https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/, 2020.

S. Team, Convolutional Neural Networks (CNN): Step 4 - Full Connection - Blogs SuperDataScience - Big Data | Analytics Careers | Mentors | Success, https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection, 2018.

J. Durán, Técnicas de Regularización Básicas para Redes Neuronales, https://medium.com/metadatos/t %C3 %A9cnicas-de-regularizaci %C3 %B3n-b %C3 % A1sicas-para-redes-neuronales-b48f396924d4:∼:text=Normalizaci %C3 %B3n %20 por %20lotes %20(Batch %20normalization)text=La %20normalizaci %C3 %B3n %20en % 20lotes %20consiste,normalizar %20las %20activaciones %20de %20salida, 2019a.

V. Rodríguez, Dropout y Batch Normalization, https://vincentblog.xyz/posts/dropout-y-batch-normalization, 2018.

D. Sood, Backpropagation concept explained in 5 levels of difficulty, https://medium.com/coinmonks/backpropagation-concept-explained-in-5-levels - of-difficulty-8b220a939db5, 2018.

M. Figueres-Moreno, Introducción-Redes-Neuronales-ArtificialesMFM.pdf, https://optimizacionheuristica.blogs.upv.es/files/2013/04/Introducci %C3 %B3nRedes-Neuronales-ArtificialesMFM.pdf, 2013.

J. Durán, Todo lo que Necesitas Saber sobre el Descenso del Gradiente Aplicado a Redes Neuronales, https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso -del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78, 2019b.

M. A, Raspberry Pi Sending data to ThingSpeak | Simplest Raspberry Pi IOT Project, https://electronicshobbyists.com/raspberry-pi-sending-data-to-thingspeak - simplest-raspberry-pi-iot-project/, 2018.

P. S. Foundation, urllib.request — Extensible library for opening URLs — Python 3.8.4rc1 documentation, https://docs.python.org/3/library/urllib.request.htmlurllib.request.Request, 2020.

I. Mebsout, Convolutional Neural Networks' mathematics | The Startup, https://medium.com/swlh/convolutional-neural-networks-mathematics-1beb3e6447c0, 2020.