Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Departamento de Matemática

PROPUESTA DE UN SISTEMA DE CALIFICACIÓN BASADO EN REDES NEURONALES

Flor de María Pérez Medina
Asesorada por William Roberto Gutiérrez Herrera

Guatemala, noviembre de 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

PROPUESTA DE UN SISTEMA DE CALIFICACIÓN BASADO EN REDES NEURONALES

TRABAJO DE GRADUACIÓN
PRESENTADO A LA JEFATURA DEL
DEPARTAMENTO DE MATEMÁTICA
POR

FLOR DE MARÍA PÉREZ MEDINA
ASESORADA POR WILLIAM ROBERTO GUTIÉRREZ HERRERA

AL CONFERÍRSELE EL TÍTULO DE
LICENCIADA EN MATEMÁTICA APLICADA

GUATEMALA, NOVIEMBRE DE 2024

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
ESCUELA DE CIENCIAS FÍSICAS Y MATEMÁTICAS

CONSEJO DIRECTIVO INTERINO

Director	M.Sc. Jorge Marcelo Ixquiac Cabrera
Representante Docente	Arqta. Ana Verónica Carrera Vela
Representante Docente	M.A. Pedro Peláez Reyes
Representante de Egresados	Lic. Urías Amitaí Guzmán García
Representante de Estudiantes	Elvis Enrique Ramírez Mérida
Representante de Estudiantes	Oscar Eduardo García Orantes
Secretario	M.Sc. Freddy Estuardo Rodríguez Quezada

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

Director	M.Sc. Jorge Marcelo Ixquiac Cabrera
Examinador	Lic. José Carlos Alberto Bonilla Aldana
Examinador	Lic. Hugo Allan García Monterrosa
Examinador	Lic. Ronald Oliverio Chubay Gallina
Secretario	MSc. Edgar Damián Ochoa Hernández

Ref. D.DTG. 013-2024
Guatemala, 11 de noviembre de 2024

El Director de la Escuela de Ciencias Físicas y Matemáticas de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del jefe de la Licenciatura en Matemática Aplicada, al trabajo de graduación titulado: "**PROPUESTA DE UN SISTEMA DE CALIFICACIÓN BASADO EN REDES NEURONALES**", presentado por la estudiante universitaria, Flor de María Pérez Medina, autoriza la impresión del mismo.

IMPRÍMASE.

"ID Y ENSEÑAD A TODOS"

M.Sc. Jorge Marcelo Ixquiac Cabrera
Director

166-2024
JMIC/Paola

AGRADECIMIENTOS

Dios	Porque solo de Él me viene la vida.
Mis padres	Flor Medina y Roberto Pérez, por su inmenso amor y dedicación, ustedes son mi modelo a seguir como personas y profesionales.
Mi esposo	José Ligorría, por ser el pilar de mi vida.
Mis catedráticos	Y en particular a mi asesor, el Lic. William Gutiérrez, por compartir sin recelo sus conocimientos y experiencias académicas.
Mi familia y amigos	Por su apoyo incansable y su fe en mí.
Universidad de San Carlos de Guatemala	Porque la educación pública hizo que la ciencia fuera accesible para mí.
Al catedrático, auxiliar y estudiantes del curso de Cálculo 1 y 2 del segundo semestre 2021	Ronald Chubay, Daniel Tobar por su disponibilidad y apoyo, y a los estudiantes por ser la base de mi investigación.

DEDICATORIA

Mis padres	Esto es por y para ustedes.
José Ligorría	Mi matemático favorito.
La comunidad de Mujeres ECFM	A las que estuvieron antes de mí, las que estuvieron conmigo y a las que vendrán.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$:=$	es definido por
$\Leftrightarrow$	si y sólo si
$\varnothing$	conjunto vacío
x	vector fila
x^T	vector columna
$\mathbf{M}$	matriz
$\mathbf{M}^\top$	matriz transpuesta
$w_j^{(i)}$	i-esimo peso del vector j
$\nabla f(\mathbf{x})$	gradiente de una función
$w_j^{(i)}$	pesos de la red neuronal
$n_j^{(i)}$	regla de propagación
$f_j^{(i)}$	función de activación
$\hat{y}^{(i)}$	valor predicho
$e_j^{(i)}$	función de error

OBJETIVOS

General

Diseñar e implementar una herramienta de calificación basada en redes neuronales que identifique las respuestas en tareas hechas a mano.

Específicos

1. Establecer los fundamentos matemáticos necesarios para el estudio de redes neuronales.
2. Describir la estructura y el funcionamiento de las redes neuronales.
3. Comprender específicamente las redes neuronales convolucionales y su aplicación en la clasificación de imágenes.
4. Plantear el diseño del sistema de calificación basado en una red neuronal convolucional y evaluar su eficiencia.

INTRODUCCIÓN

Debido a la pandemia de COVID19 declarada en marzo de 2020 en Guatemala y la suspensión de actividades presenciales, gran parte de la educación tuvo que migrar al entorno virtual, potenciando el uso de plataformas educativas en línea, por ejemplo Moodle o Google Classroom. Dado que se hizo de forma improvisada, por falta de formación en educación a distancia, esto llevó a la necesidad de requerir tareas en formato digital, que en principio están en un soporte análogo (hojas de papel), tuvieron que trasladarse a un soporte digital (imágenes jpg, png, pdf), ya sea por el uso de un escáner o por el uso de la cámara fotográfica de un celular o tableta, y la consiguiente calificación de éstos dentro de la plataforma o su descarga.

Si en formato análogo se debe calificar una pila de tareas, la improvisación descrita, provoca que se debe calificar una pila de tareas digitales, una a una, lo que para un gran volumen de alumnos o tareas asignadas, es un gran esfuerzo de horas de trabajo. Dado que las herramientas tecnológicas buscan facilitar o reducir el trabajo físico de las personas, el presente trabajo de graduación propone una solución basada en **inteligencia artificial (IA)** en el ámbito de las *redes neuronales* que clasifique las respuestas en tareas hechas a mano. Y así, tener un sistema automatizado de calificación.

Para la construcción de este proyecto, se planteo un plan de trabajo, dividido en un trabajo de campo y otro de gabinete. En el trabajo de campo se incluye trabajar con los estudiantes de *Cálculo 1 y 2* en modalidad virtual de la **Escuela de Ciencias Físicas y Matemáticas (ECFM)** en el segundo semestre del 2021, con ellos se realizaron las pruebas tanto para la clasificación de imágenes como para calificar tres de las tareas del curso utilizando el sistema de calificación propuesto. El trabajo de gabinete incluye la revisión bibliográfica acerca de IA, de las mejores técnicas para el procesamiento de imágenes, reconocimiento de formas, lenguajes de programación, plataformas de implementación.

Este trabajo consta de cuatro partes, la primera la presentación de los fundamentos matemáticos necesarios para el estudio de las redes neuronales, el segundo

la descripción de todos los componentes y el funcionamiento de las redes neuronales, el tercero describe a detalle la arquitectura de las redes neuronales convolucionales, y el último presenta los datos utilizados, la arquitectura del modelo, el diseño del sistema de calificación propuesto, y los resultados de probar tanto el modelo de clasificación como el sistema de calificación.

1. CONCEPTOS PRELIMINARES

1.1. Lógica Proposicional

Una **proposición** a una oración o enunciado que podemos determinar si es verdadero o falso. Una proposición es **simple** cuando ninguna otra de sus partes es a su vez una proposición, las podemos representar con las letras **p**, **q** y **r**. Una proposición es **compuesta** cuando puede descomponerse en varias proposiciones simples. De manera general denotamos con **1** al valor verdadero o *true* y con 0 al valor falso o *false*. Podemos combinar las proposiciones simples a través de los siguientes **conectivos lógicos**:

1. **Conjunción**: Se denota como $p \wedge q$ y se lee «p y q». La condición que hace una conjunción verdadera, es que ambos componentes conjuntivos sean verdaderos, en caso contrario la conjunción es falsa.

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

2. **Disyunción:** Se denota como $p \vee q$ y se lee «p o q». La condición para que una disyunción sea verdadera, es que al menos una de las componentes sea verdadero.

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

3. **Implicación:** Se denota como $p \to q$ y se lee «p implica q», a p le llamamos antecedente y a q el consecuente. La única condición que hace que la implicación sea falsa, es que el antecedente p sea verdadero y el consecuente q falso.

p	q	$p \to q$
1	1	1
1	0	0
0	1	1
0	0	1

4. **Bicondicional**: Se denota como $p \leftrightarrow q$ y se lee «p si y solo si q». La condición que hace que una proposición bicondicional sea verdadera, es que ambas proposiciones tengan el mismo valor de verdad.

p	q	$p \leftrightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

Decimos que una proposición es *abierta* si depende de un valor asignado para determinar si es verdadera o falsa. De otra manera una frase declarativa es una proposición abierta si:

- Contiene una o más variables.
- Se convierte en una proposición al sustituir las variables por opciones válidas.

1.2. Geometría y álgebra lineal

1.1 Definición. Definimos un **vector columna** como un arreglo de n números reales como

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Los números x_i representan la i-ésima componente del vector x, con $i = 1, \ldots, n$.

Definimos también un **vector fila** como

$$[x_1, x_2, \ldots, x_n]$$

el vector **transpuesto** de un vector columna dado y se denota como x^T. Dado un vector, la primera interpretación que debemos darle es como un punto en el espacio. En dos o tres dimensiones, podemos visualizar estos puntos utilizando los componentes de los vectores para definir la ubicación de los puntos en el espacio en comparación con una referencia fija llamada origen.

En un **espacio euclídeo** ordinario los vectores se representan como segmentos orientados entre puntos de dicho espacio. La **norma** de un vector nos dice qué tan grande es un vector, la noción de tamaño que se considera aquí no se refiere a la dimensión, sino a la magnitud de los componentes. Supongamos que los elementos del vector n-dimensional $\mathbf{x}$ son $x_1, \ldots, x_n$. Definimos la norma $\|\cdot\|_1$ que se expresa como la suma de los valores absolutos de los elementos del vector:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \,.$$

La norma $\|\cdot\|_2$ de $\mathbf{x}$ es la raíz cuadrada de la suma de los cuadrados de los elementos vectoriales:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

donde el subíndice 2 frecuentemente se omite en las normas $\|\mathbf{x}\|_2$, es decir, $\|\mathbf{x}\|$ es equivalente a $\|\mathbf{x}\|_2$.

Sean dos vectores $a = [a_1, a_2, \ldots, a_n]^T$ y $b = [b_1, b_2, \ldots b_n]^T$. Definimos el **producto punto** de a y b como el número $a \cdot b$ dado por

$$a \cdot b = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n.$$

Así, para hallar el producto punto de a y b se multiplican las componentes correspondientes y se suman. El resultado no es un vector, es un número real

$$\mathbf{u}^\top \mathbf{v} = \sum_i u_i \cdot v_i.$$

El producto punto también admite una interpretación geométrica, está estrechamente relacionado con el ángulo entre dos vectores. Consideremos dos vectores

específicos,

$$\mathbf{v} = (r, 0) \quad \text{y} \quad \mathbf{w} = (s\cos(\theta), s\,\text{sen}(\theta)).$$

El vector $\mathbf{v}$ es de longitud r y corre paralelo al eje x, y el vector $\mathbf{w}$ es de longitud s y forma un ángulo θ con el eje x. Si calculamos el producto escalar de estos dos vectores, tenemos que

$$\mathbf{v} \cdot \mathbf{w} = rs\cos(\theta) = \|\mathbf{v}\|\|\mathbf{w}\|\cos(\theta)$$

de donde,

$$\theta = \text{arc}\cos\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}\right).$$

1.2 Definición. Podemos definir una **matriz** como un arreglo de escalares en filas y columnas. Usamos $\mathbf{A} \in \mathbb{R}^{m\times n}$ para denotar a la matriz $\mathbf{A}$ que consta de m filas y n columnas de escalares que pertenecen a los reales. La escribimos de esta manera, donde cada elemento a_{ij} pertenece a la i-ésima fila y la j-ésima columna:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Cuando intercambiamos filas y columnas de una matriz, el resultado se denomina transposición de la matriz. Formalmente escribimos la transpuesta de la matriz $\mathbf{A}$ como $\mathbf{A}^\top$ y si $\mathbf{B} = \mathbf{A}^\top$ entonces $b_{ij} = a_{ji}$ para cualquier i y j. Por tanto, la transpuesta de $\mathbf{A}$ en es una matriz de $n \times m$:

$$\mathbf{A}^\top = \begin{bmatrix} a_{11} & a_{21} & \ldots & a_{m1} \\ a_{12} & a_{22} & \ldots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{mn} \end{bmatrix}.$$

Al igual que los vectores, las matrices se pueden sumar solo si tienen exactamente el mismo tamaño. Por ejemplo, se pueden sumar las matrices $\mathbf{A}$ y $\mathbf{B}$ sólo si $\mathbf{A}$y $\mathbf{B}$ tienen exactamente el mismo número de filas y columnas. La (i, j)-ésima entrada de $\mathbf{A}+\mathbf{B}$es la suma de las(i, j)-ésimas entradas de $\mathbf{A}$ y $\mathbf{B}$, respectivamente. La suma de matrices es conmutativa, porque hereda la propiedad conmutativa de la

suma de escalares de sus entradas individuales.

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}.$$

Una matriz $\mathbf{A}$ de $n \times m$ puede multiplicarse por un vector de columna x de dimensión m como $\mathbf{A}x$, o se puede multiplicar con un vector fila n-dimensional y como $y\mathbf{A}$. Cuando una matriz $\mathbf{A}$ de dimensión $n \times m$ se multiplica con el vector de columna m-dimensional x para crear $\mathbf{A}x$, la multiplicación se realiza entre los m elementos de cada fila de la matriz $\mathbf{A}$ y los m elementos del vector columna x, y luego estos productos por elementos se agregan para crear un escalar. Al final del proceso, se calculan n escalares y se ordenan en un vector de columna n-dimensional en el que el i-ésimo elemento es el producto entre la i-ésima fila de $\mathbf{A}$ y x. Por ejemplo,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

Podemos notar que esta operación es la misma que el producto escalar, excepto que es necesario transponer las filas de $\mathbf{A}$ a los vectores de columna para expresarlo rigurosamente como un producto escalar. La multiplicación de una matriz $\mathbf{A}$ de $n \times m$ con un vector de columna x de dimensión m para crear un vector columna n-dimensional $\mathbf{A}x$ se interpreta a menudo como una transformación lineal de espacio m-dimensional al espacio n-dimensional.

1.3 Definición. Sean V y W espacios vectoriales, una **transformación lineal** T de V en W en $\mathbb{R}$ es una aplicación lineal $T\colon V \to W$ si para todo par de vectores $u, v \in V$ y para todo escalar $k \in \mathbb{R}$ se cumple:

1. $T(u + v) = T(u) + T(v)$
2. $T(ku) = kT(u)$

1.4 Definición. Sean V, W espacios vectoriales de dimensiones finitas sobre un campo F, sea $\mathcal{A} = (a_1, \cdots, a_n)$ una base de V, sea $\mathcal{B} = (b_1, \ldots, b_m)$ una base de W, y sea $T \in \mathcal{L}(V, W)$. La matriz de T en bases B y A (o matriz asociada con T respecto a las bases $\mathcal{B}$ y $\mathcal{A}$), denotada por $\mathcal{T}_{\mathcal{B},\mathcal{A}}$, se define como la matriz cuyas columnas son columnas de coordenadas de los vectores $T(a_1), \cdots, T(a_n)$ en base $\mathcal{B}$:

$$\mathcal{T}_{\mathcal{B},\mathcal{A}} = [(T(a_1))_{\mathcal{B}} \cdots (T(a_n))_{\mathcal{B}}]$$

1.5 Definición. Sean $u_1, u_2, \ldots, u_n \in \mathbb{R}$ con al menos un i tal que $u_i \neq 0$. Al conjunto de los $x = (x_1, \cdots, x_n)^t$ tales que,

$$u_1x_1 + \ldots + u_nx_n = v.$$

Le llamamos **hiperplano** de $\mathbb{R}^n$, y lo describimos como, $H(u, v) = \{x \in \mathbb{R}^n : u^Tx = v\}$ donde $u = (u_1, \cdots, u_n)^T$.

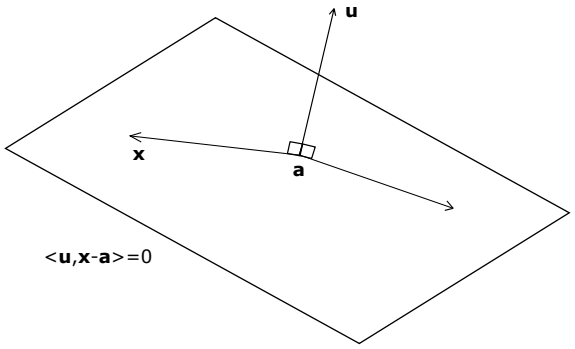


Figura 1.1. El hiperplano $H = \{x \in \mathbb{R}^n : u^T(x - a) = 0\}$. Fuente: Elaboración propia.

Un hiperplano no es necesariamente un subespacio de $\mathbb{R}^n$, ya que, en general no contiene al origen. Al trasladar un hiperplano para que contenga al origen de $\mathbb{R}^n$ se vuelve un subespacio de $\mathbb{R}^n$. Como la dimensión de este subespacio es $n - 1$, decimos que la dimensión del hiperplano es de $n - 1$.

El hiperplano $H = \{x : u_1x_1 + \cdots + u_nx_n = v\}$ divide a $\mathbb{R}^n$ en dos semi-espacios. Uno de estos semiespacios consiste en todos los puntos que satisfacen la desigualdad $u_1x_1 + u_2x_2 + \ldots + u_nx_n \geq v$ denotada por

$$H_+ = \{x \in \mathbb{R}^n : u^Tx \geq v\}.$$

El otro semi-plano consiste en todos los puntos que satisfacen la desigualdad $u_1x_1 + u_2x_2 + \cdots + u_nx_n \leq v$ denotada por

$$H_+ = \{x \in \mathbb{R}^n : u^Tx \leq v\}.$$

1.3. Elementos de cálculo

1.6 Definición. Supongamos que tenemos una función $f\colon \mathbb{R} \to \mathbb{R}$, cuya entrada y salida son ambas escalares. La **derivada** de f se define como

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

si existe este límite.

Si $f'(a)$ existe, se dice que f es diferenciable en a. Si f es diferenciable en cada número de un intervalo, entonces esta función es diferenciable en este intervalo. Podemos interpretar la derivada $f'(a)$ como la tasa de cambio instantánea de $f(x)$ con respecto a x. La denominada tasa de cambio instantánea se basa en la variación h en x, que se acerca a 0.

Dado $y = f(x)$, donde x y y son la variable independiente y la variable dependiente de la función f, respectivamente. Las siguientes expresiones son equivalentes:

$$f'(x) = y' = \frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx} f(x) = Df(x) = D_x f(x).$$

1.7 Definición. Si g es una función derivable en x y f es una función diferenciable en $g(x)$, la función composición $f(g(x))$ es derivable en x y su derivada es:

$$f'\left(g(x)\right) = f'\left(g(x)\right) \cdot g(x)$$

1.8 Definición. Sea $y = f(x_1, x_2, \ldots, x_n)$ una función con n variables. La **derivada parcial** de y con respecto a su i-ésimo parámetro x_i es

$$\frac{\partial y}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}.$$

Para calcular $\frac{\partial y}{\partial x_i}$, simplemente podemos tratar $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ como constantes y calcular la derivada de y con respecto a x_i. Podemos concatenar derivadas parciales de una función multivariante con respecto a todas sus variables para obtener el vector gradiente de la función. Supongamos que el valor de entrada de la función $f\colon \mathbb{R}^n \to \mathbb{R}$ es un vector n-dimensional $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$ y el valor de salida es un escalar.

1.9 Definición. El **gradiente** de la función $f(x)$ con respecto a $\mathbf{x}$ es un vector de

n derivadas parciales:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^\top.$$

Las **curvas de nivel** de una función f de varias variables son las curvas cuyas ecuaciones son $f(x_1, x_2, \ldots x_n) = k$, donde k es una constante en el rango de f. Es decir, una curva de nivel es el conjunto de todos los puntos en el dominio de f en el cual f toma un valor dado k.

1.10 Definición. Decimos que una función es *convexa* si cumple las siguientes propiedades para cualquier par de vectores x_1 y x_2 y cualquier escalar $\lambda \in (0, 1)$:

$$f\left(\lambda x_1 + (1-\lambda)x_2\right) \leq \lambda f\left(x_1\right) + (1-\lambda)\, f\left(x_2\right)$$

Una función continuamente diferenciable de una variable es convexa en un intervalo si y solo si la función se encuentra por encima de todas sus tangentes:

$$f\left(x_2\right) \geq f\left(x_1\right) + f'\left(x_1\right)\left(x_2 - x_1\right)$$

para todo x_1 y x_2 en el intervalo. En particular, si $f'\left(c\right) = 0$, entonces c es un mínimo absoluto de $f(x)$.

1.4. Elementos de teoría de grafos

1.11 Definición. Sea V un conjunto finito no vacío, y sea $E \subseteq V \times V$. El par (V, E) es un **grafo dirigido** (sobre V), donde V es el conjunto de *vértices* o *nodos* y E es su conjunto de *aristas*. Escribimos $G = (V, E)$ para denotar al grafo.

Un grafo no dirigido es un grafo $G = (V, E)$ donde V es no vacío y E es un conjunto de pares no ordenados de elementos de V.

En la figura tenemos un grafo dirigido sobre $V = \{a, b, c, d, e, f\}$ con $E = \{(a, b), (a, e), (b, c), (b, e), (c, d), (d, e), (d, f)\}$. La dirección de una arista se indica al colocar una flecha sobre ella. Para cualquier arista, decimos que la arista (b, c) es **incidente** con los vértices b y c, el vértice b es **adyacente hacia** c, y c **adyacente desde** b. Tenemos que para la arista (c, d), c es el origen y d es el término o vértice terminal.

Cuando no importa la dirección de las aristas, la estructura $G = (V, E)$, donde E es un conjunto de pares no ordenados sobre V, decimos que es un **grafo no dirigido** .

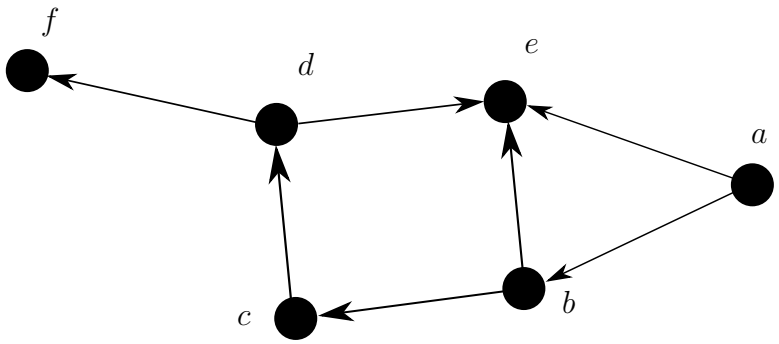


Figura 1.2. Grafo dirigido.

1.12 Definición. Sean x, y vértices de un grafo no dirigido $G = (V, E)$. Un **camino** $x - y$ en G es una sucesión alternada finita

$$x = x_0, e_1, x_1 e_2, x_2, e_3, \ldots, e_{n-1}, x_{n-1}, e_n, x_n = y$$

de vértices y aristas de G, que comienza en el vértice x y termina en el vértice y y que contiene las n aristas $e_i = \{x_{i-1}, x_i\}$ donde $1 \leq i \leq n$.

1.13 Definición. Sea $G = (V, E)$ un grafo no dirigido. Decimos que G es **conexo** si existe un camino simple entre cualesquiera dos vértices distintos de G.

1.14 Definición. Un grafo $G = (V, E)$ es un **multigrafo** si existen $a, b \in V, a \neq b$, con dos o más aristas de la forma (a, b) para grafo dirigido o $\{a, b\}$ para grafo no dirigido.

1.5. Aprendizaje automático

Es una rama de la Inteligencia Artificial que proporciona a los sistemas la capacidad de aprender y mejorar automáticamente de la experiencia sin ser programados explícitamente [2]. El proceso de aprendizaje comienza con observaciones o datos, como ejemplos, estímulo directa o instrucción, con el fin de buscar patrones en los datos y tomar mejores decisiones en el futuro con base en los ejemplos proporcionados.

Los algoritmos de aprendizaje automático se clasifican como supervisados o no supervisados.

- **Aprendizaje supervisado** El objetivo estos algoritmos es simular la toma de desiciones basadas en estímulos que realiza el cerebro humano, a esta si-

mulación se le llama Aprendizaje Supervisado. El algoritmo de aprendizaje puede comparar su salida con la salida prevista correcta y encontrar errores para modificar el modelo adecuadamente.

Los tipos de tarea de aprendizaje supervisado se eligen dependiendo del problema que se quiera resolver, siendo estos de clasificación cuando $y^{(i)}$ es binario o discreto y de regresión cuando es continuo.

  - **Clasificación** Es un proceso de búsqueda de una función que ayuda a dividir el conjunto de datos en clases en función de diferentes parámetros. En los problemas de clasificación, un programa de computadora se entrena en el conjunto de datos de entrenamiento y, en base a ese entrenamiento, categoriza los datos en diferentes clases.
    La tarea del algoritmo de clasificación es encontrar la función para mapear la entrada x al valor de salida discreto y.
  - **Regresión** La regresión es un proceso de encontrar las correlaciones entre variables dependientes e independientes para ayudar a predecir las variables continuas.
    La tarea del algoritmo de regresión es encontrar la función para mapear la entrada x al valor de salida continuo y.

- **Aprendizaje no supervisado**: Estos algoritmos se utilizan cuando la información que se usa para entrenar no está clasificada ni etiquetada. estudia cómo los sistemas pueden inferir una función para describir una estructura oculta a partir de datos no etiquetados. El sistema no encuentra el resultado correcto, pero explora los datos y puede extraer inferencias de conjuntos de datos para describir estructuras ocultas a partir de datos no etiquetados.

2. REDES NEURONALES

2.1. Contexto biológico

Las neuronas son las células del sistema nervioso, y posee todos los elementos de una célula biológica. A su vez, las neuronas tienen características propias, estas sirven para transmitir y emitir información. Las redes de neuronas son aquellas que comunican mediante impulsos nerviosos la información desde el cerebro hacia el resto del cuerpo o viceversa. Una neurona tiene tres partes principales:

- **Dendritas:** forman pequeñas prolongaciones ramificadas que salen de las diferentes partes del cuerpo celular, estas reciben estímulos de entrada.
- **Cuerpo de la neurona:** en este espacio es donde se sintetiza o genera la mayor parte de las moléculas de la neurona, aquí se procesan los estímulos de entrada.
- **Axón:** se encarga de transmitir las señales eléctricas desde el cuerpo de la neurona hasta los botones terminales, es decir emiten estímulos de salida a las dendritas de otras neuronas.

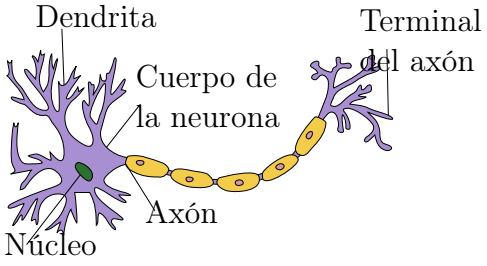


Figura 2.1. Estructura de una neurona. Fuente: imagen tomada de Quasar Jarosz de Wikipedia en inglés, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=7616130`, con modificaciones hechas para este trabajo.

Las dendritas se ramifican formando una red, mientras que el axón se ramifica en filamentos mediante los cuales establece conexión con los cuerpos de otras

neuronas, a esta conexión se le llama **sinapsis**. Las sinapsis liberan sustancias químicas transmisoras y entran a la dentrita, con lo cual se eleva el potencial eléctrico del cuerpo de la célula. Cuando llegan a su límite, se envía al axón un **impulso eléctrico**. El impulso se transmite a través de las ramas del axón llegando a la sinapsis y liberando transmisores en los cuerpos de otras neuronas. Las sinapsis que aumentan el potencial se conocen como **excitadoras** y las que los disminuyen como **inhibidoras**.

El proceso de aprendizaje de una neurona se resume de la siguiente manera: las dendritas se encargan de captar los impulsos nerviosos que emiten otras neuronas. Estos impulsos, se procesan en el cuerpo de la neurona y se transmiten a través del axón que emite un impulso nervioso hacia las neuronas contiguas.

2.2. Redes neuronales artificiales

Las **redes neuronales artificiales** (RNA) son modelos matemáticos que intentan reproducir el comportamiento del cerebro humano para la toma de desiciones basadas en estímulos. Una red neuronal artificial está compuesta por un conjunto de capas donde la información se encuentre distribuida a lo largo de las sinapsis de la red. Podemos distinguir tres tipos de capas:

- Las neuronas de entrada representan las señales que son capturadas por las dentritas que entran a la sinapsis, cada neurona está conectada con otras donde la señal es ponderada a través de pesos.

- Las neuronas ocultas reciben estímulos y transmiten salidas son las encargadas de procesar la información a través de pesos. Los pesos representan la intensidad de la sinapsis que conecta dos neuronas. Estos pesos pueden incrementar o inhibir el estado de activación de las neuronas contiguas.

- Las neuronas de salida reciben la información procesada y emiten los valores de salida que son dados por una función que modifica el valor o lo limita antes de transmitirlo a otra neurona. A esta función se le llama función de activación.

Podemos definir una red neuronal como un grafo dirigido con las siguientes propiedades:

1. A cada nodo i se le asocia un variable de estado.

2. A cada conexión (i, j) de los nodos i y j se el asocia un peso.

3. A cada nodo i se le asocia un umbral.

4. Para cada nodo i se define una función, que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos j a él conectados. Esta función proporciona el nuevo estado del nodo.

2.2.1. Arquitectura de una RNA

Se refiere a la organización de las capas establecidas de acuerdo al problema a resolver. Los parámetros a tomar en cuenta son números de capas, cantidad de neuronas por capa, tipo de conexión entre las neuronas, este último se refiere a la manera en que se distribuye la información a través de las neuronas [1].

Es una topología ya que todos los grafos de una red neuronal con parámetros establecidos son equivalentes entre sí y representan al mismo objeto en una topología de grafos. Cuando se realiza una clasificación de la red en términos topológicos, se suele distinguir entre las redes con una sola capa o nivel de neuronas y las redes con múltiples capas.

Algunas de las arquitecturas más utilizadas de redes neuronales son:

1. Perceptrón: Es modelo matemático más simple de una red neuronal, toma los valores de entrada, los sintetiza, aplica la función de activación y los pasa a la capa de salida.

2. Red neuronal hacia adelante (*Feed Foward Network*): Se caracterizan por la dirección del flujo de información entre sus capas, la información fluye en una sola dirección, hacia adelante, desde las neuronas de entrada pasando por las neuronas ocultas hacia las neuronas de salida sin bucles ni ciclos.

3. Red neuronal recurrente (*Recurrent Neural Network*): Esta red presenta un nuevo tipo de neuronas, las neuronas recurrentes tienen conexiones entre pases, conexiones a través del tiempo. Las neuronas reciben información no solo de la capa anterior, sino también de ellas mismas de la pasada anterior, esto significa que el orden en que alimenta la entrada y entrena la red es importante. Este tipo de red se usa principalmente cuando el contexto es importante, cuando las decisiones de iteraciones o muestras pasadas pueden influir en las actuales.

4. Red neuronal convolucional (*Convolutional Neural Network*): Estas redes presentan capas de convolución, de agrupación y núcleos, cada uno con un propósito diferente, las neuronas de convolución procesan los datos de entrada

y las capas de agrupación los simplifican a través de funciones no lineales, lo que reduce las funciones innecesarias. Se utilizan para el reconocimiento de imágenes, operan en un pequeño subconjunto de imágenes donde la capa de entrada se desliza a lo largo de la imagen, píxel por píxel, los datos se pasan a capas de convolución, formando un embudo.

2.2.2. Componentes de una RNA

1. **Valores de entrada** $x^{(i)}, y^{(i)}$. Son las variables que comunican a la red neuronal con el mundo exterior. Podemos clasificarlas en:

   1.1 Discretas: cuando la variable toma una cantidad entera de valores.

   1.2 Continuas: cuando la variable toma valores en un rango numérico.

2. **Pesos** $w_j^{(i)}$. Representan la conexión sináptica que sucede en las neuronas biológicas, estos son coeficientes que se modifican en respuesta al entrenamiento de acuerdo a la regla de propagación y pertenecen a las neuronas ocultas. Los pesos pueden ser positivos, negativos o nulo. Cuando es positivo actúa como excitador, en el caso de que sea negativo el peso actúa como inhibidor y cuando el peso es cero quiere decir que no hay comunicación entre el par de neuronas.

3. **Regla de propagación** $n_j^{(i)}$. Integra toda la información de entrada e indica como debe ser transformada capa a capa a través de las neuronas ocultas para llegar a la neurona de salida.

   $$n_j^{(i)}\big(w_j^{(i-1)}, a_j^{(i)}\big)$$

   La regla de propagación más utilizada es la sumatoria de todas las entradas multiplicadas por los pesos de las conexiones.

   $$n_j^{(i)}\big(w_j^{(i-1)}, a_j^{(i)}\big) = \big\langle w_j^{(i-1)}, a_j^{(i)} \big\rangle.$$

4. **Función de activación** $f_j^{(i)}$. Determina el nivel de activación de la neurona es decir, mapea el valor obtenido de la propagación de la capa anterior según la regla definida. Al estado de activación, es decir a la propagación de la capa $i-1$ sobre la capa i en la neurona j se denota como $a_j^{(i)}$.

   $$a_j^{(i)} = f_j^{(i)}\left(n_j^{(i)}\big(w_j^{(i-1)}, a_j^{(i)}\big)\right).$$

Las funciones de activación más utilizadas en los distintos modelos de redes neuronales son:

- Función lineal:

$$f(x) = ax, \quad a \in \mathbb{R}.$$

  Sin importar el número de capas, si todas son de naturaleza lineal, la función de activación final de la última capa no es más que una función lineal de la entrada de la primera capa. Se utiliza nada más en la capa de salida y su rango es de $(-\infty, \infty)$.

- Función escalón:

$$f(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

  Es una de las funciones de activación más sencillas. Consideramos un valor de umbral y si el valor de la entrada neta es mayor que el umbral, entonces la neurona se activa. El rango de está función está en $(-\infty, \infty)$.

- Sigmoidal

$$f(x) = \frac{1}{1 + e^{-x}}.$$

  Es una función no lineal. Los valores de x se encuentran entre -2 y 2, los valores de $f(x)$ son muy pronunciados. Esto significa que pequeños cambios en x también provocarían grandes cambios en el valor de $f(x)$. Se usa en la capa de salida de una clasificación binaria, donde el resultado es 0 o 1, ya que el valor de la función sigmoidea se encuentra solo entre 0 y 1.

- Tangente hiperbólica

$$f(x) = \tanh x.$$

  Es una función similar a la Sigmoide pero centrada en 0 y con rango entre $[-1, 1]$. Además, es una función continua. La ventaja es que las entradas negativas se mapearán estrictamente negativas y las entradas cero se mapearán cerca de cero. Esto permite a centrar los datos acercando la media a 0.

- ReLu

$$f(x) = \text{máx}(0, x).$$

  ReLu por sus siglas en inglés (*Rectified Linear Units*). Es la función de activación más utilizada,devuelve el que sea mayor entre cero o el valor de

entrada. Es decir, si x es negativo, devolverá cero. ReLU nunca devolverá un valor negativo, por lo que debe aplicarse a las salidas de las capas ocultas para que sean números positivos consistentes.

- Softmax
$$f_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$
La función softmax también es un tipo de función sigmoidea, pero es útil cuando tratamos de manejar problemas de clasificación. Softmax devuelve valores entre 0 y 1, genera un vector de probabilidades y se asegura de que todo el vector de resultados sume 1. Se utiliza como capa final en los modelos de clasificación.

5. **Valores de salida**
$$\hat{y}^{(i)} = F\big(n_j^{(i)}\big).$$
Es el valor estimado dado por la función de activación de la capa de salida.

6. **Función de error** $e_j^{(i)}$. Es la función que mide la diferencia entre la el resultado de salida de la red y la salida correcta. El objetivo de la función de costo es minimizar la diferencia entre estos dos valores. Existen distintas funciones de error que se eligen según la tarea que se este resolviendo. Algunas de estas son:

   - Error cuadrático:
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2.$$
Es el promedio de la diferencia al cuadrado entre el valor predicho y el valor real.

   - Error de entropía cruzada
$$\text{CCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log\left(p_{ij}\right)$$
La entropía cruzada categórica calcula la suma del producto de la etiqueta verdadera y y el logaritmo de la probabilidad predicha p para cada clase y luego niega el resultado. La pérdida es alta cuando las predicciones del modelo son confiables pero incorrectas, y baja cuando las predicciones del modelo coinciden con las etiquetas verdaderas.

2.3. Aprendizaje de una RNA

El proceso de aprendizaje de una red neuronal artificial se basa en el funcionamiento de una red neuronal biológica, en este proceso la red actualiza los pesos y, en algunos casos, la estructura con el propósito de que la red pueda llevar a cabo de forma efectiva una tarea determinada.

Una neurona obtiene sus entradas directamente de los datos que nos interesan o de las salidas de otras neuronas. Estas entradas son como una lista, y cada elemento de la lista representa una característica diferente de los datos.

Para cada entrada, la neurona hace algunos cálculos: multiplica la entrada por un peso y luego agrega un sesgo o *bias*. Los pesos son la forma que tiene la neurona de decidir qué tan importante es una entrada, y en el sesgo es un ajuste para asegurarse de que la salida de la neurona se ajuste perfectamente. Durante el entrenamiento de la red, esta ajusta estos pesos y sesgos para mejorar su trabajo. Luego de esto, la neurona suma todas estas entradas y sesgos ponderados y ejecuta el total a través de la función de activación.

2.3.1. Entrenamiento de una RNA

El proceso de aprendizaje de una red neuronal busca un conjunto de pesos que permitan a la red desarrollar una tarea determinada. Es un proceso iterativo, en el cual se va refinando la solución hasta alcanzar un nivel de operación óptimo. La mayoría de los métodos de entrenamiento proponen una función de error que mide el rendimiento de la red en función de los pesos. El objetivo del método es encontrar el conjunto de pesos que minimizan o maximizan la función.

Backpropagation

El algoritmo de propagación hacia atrás o *backpropagation* es uno de los más importantes de aprendizaje supervisado a la hora de entrenar una red neuronal [7]. Calcula el gradiente de una función de error con respecto a los pesos de la red para un único valor de entrada-salida, y lo hace de manera eficiente, calculando el gradiente una capa a la vez, iterando hacia atrás desde la última capa para evitar cálculos redundantes de términos intermedios utilizando la regla de la cadena. En otras palablas, el algoritmo backpropagation realiza un paso hacia atrás después de cada paso hacia adelante a través de la red, mientras ajusta los parámetros del modelo.

Parámetros:

- $x = (x_1, x_2, \ldots, x_n)$ vector de valores de entrada.
- $t = (t_1, t_2, \ldots, t_n)$ vector de valores objetivo.
- δ_k error en una neurona de salida.
- δ_j error en una capa oculta.
- α tasa de aprendizaje.
- b_{0j} sesgo de la neurona oculta j.

Algoritmo de entrenamiento

Paso 1. Iniciar con valores aleatorios pequeños para los pesos.

Paso 2. Cada unidad de entrada recibe la señal y transmite la señal x_i a todas las unidades.

Paso 3. Cada unidad oculta z_j con $j \in (1, \ldots, a)$ suma su señal de entrada ponderada para calcular su entrada neta.

$$z_{inj} = b_{0j} + \sum x_i b_{ij}, \quad (i = 1 \ldots, n).$$

Aplicando la función de activación $z_j = f(z_{inj})$ y envía esta señal a todas las unidades en la capa de salida. Para cada la salida y_k con $k \in (1, \ldots, m)$ suma sus señales de entrada ponderadas

$$y_{ink} = w_{0k} + \sum z_i w_{jk}, \quad (j = 1, \ldots, a)$$

y se aplica su función de activación para calcular las señales de salida.

$$y_k = f(y_{ink}).$$

Error de retropropagación:

Paso 4. Cada unidad de salida y_k recibe un patrón objetivo que corresponde a un patrón de entrada y se calcula el error:

$$\delta_k = (t_k - y_k) + y_{ink}.$$

Paso 5. Cada unidad oculta z_j con $j \in (1, \ldots, a)$ suma la entrada de todas las unidades en la capa superior

$$\delta_{inj} = \sum \delta_j w_{jk}.$$

Se calcula el error de la capa oculta:

$$\delta_j = \delta_{inj} + z_{inj}.$$

Actualización de los pesos y sesgos:

Paso 6. Cada unidad de salida y_k con $k \in (1, \ldots, a)$ actualiza su peso y su sesgo. El término de corrección del peso esta dado por:

$$\Delta w_{jk} = \alpha \delta_k z_j$$

y el del sesgo:

$$\Delta w_k = \alpha \delta_k$$

por lo que,

$$w_{jk(\text{actual})} = w_{jk(\text{anterior})} + \Delta w_{jk},$$
$$w_{0k(\text{actual})} = w_{0k(\text{anterior})} + \Delta w_{0k}.$$

Paso 7. Se prueba la condición de parada que puede ser la minimización del error o el número de épocas.

2.4. Aplicaciones

- Reconocimiento de patrones: El reconocimiento de patrones es un proceso de análisis de datos que utiliza algoritmos de aprendizaje automático para clasificar los datos de entrada en objetos, clases o categorías en función de patrones, características o regularidades reconocidas en los datos. Tiene diversas aplicaciones en los campos de la astronomía, la medicina, la robótica y la teledetección por satélite, entre otros.

- Visión artificial: Permite a los computadores y sistemas, a través de imágenes digitales o cualquier entrada visual, extraer información significativa y tomar

decisiones basadas en esta. Para resolver este tipo de problemas se utilizan las redes neuronales convolucionales.

- Procesamiento del lenguaje natural: Es la habilidad de un sistema para comprender el lenguaje humano tal como se habla y se escribe a través de un modelo de reglas del lenguaje humano combinado con modelos de aprendizaje automático y matemáticos que permiten a los computadores y sistemas reconozcan, comprendan y generen texto y voz.

- Predicción y toma de decisiones: Las redes neuronales son una clase potente de algoritmos de aprendizaje automático aplicados a problemas de toma de decisiones. Aprenden de los datos para crear predicciones o conclusiones basadas en patrones y relaciones intrincados. Se han convertido en una herramienta clave en la IA por su adaptabilidad y capacidad para manejar temas complejos.

3. REDES NEURONALES CONVOLUCIONALES

La primera motivación para las redes neuronales convolucionales se derivó de experimentos realizados por Hubel y Wiesel en 1959 [5], en la corteza visual de un gato. La corteza visual tiene pequeñas regiones de células que son sensibles a regiones específicas en el campo visual. Es decir, que si áreas específicas del campo visual se activan, entonces las células en esas áreas también van a activarse, éstas dependen de la forma y la orientación de los objetos en el campo visual, las células están conectadas a través de capas y a través de estos experimentos se llegó a la conclusión de que los mamíferos utilizan estas capas para construir porciones de imágenes en distintos niveles de abstracción.

Los modelos de redes neuronales convolucionales están inspirados por los procesos biológicos que tienen lugar en la corteza visual, donde las neuronas individuales responden a estímulos en un área restringida del campo visual. Este área se superpone parcialmente con el de las neuronas más próximas, cubriendo de forma colectiva el campo visual completo.

Las redes neuronales que se utilizan para tareas de clasificación tienen una fase de extracción de características. Una característica es un elemento geométrico que se identifica en una zona visual como bordes, esquinas y puntos, la extracción de características consiste en encontrar la posición de estas características. Luego se hace una reducción con las características relevantes, y se repite el proceso hasta llegar a las neuronas de clasificación con las características más importantes. Este proceso es parecido a la estimulación de las células en la corteza visual.

Una Red Neuronal Convolucional, CNN por sus siglas en inglés (*Convolutional Neural Network*), contiene varias capas ocultas especializadas y con una jerarquía, esto quiere decir que las primeras capas pueden detectar lineas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

3.1. Estructura

La tarea principal de la red neuronal convolucional es extraer, procesar y clasificar las características principales de las imágenes que recibe de entrada para esto asigna importancia (pesos) a ciertas características como líneas, puntos, curvas, etc. A medida que se avanza en las capas se va aumentando la complejidad de identificación de estás características, permitiendo que se identifique el objeto buscado.

En las redes neuronales convolucionales, cada capa se organiza en una distribución espacial de rejilla, ya que estas relaciones espaciales se heredan porque cada valor se basa en una región de la capa anterior. Estas relaciones son importantes ya que la convolución depende de estas.

Las capas que forman una red neuronal convolucional son:

- Capas convolucionales: en estas capas se da la extracción de características con los píxeles cercanos de la imagen.
- Capa de función de activación: en esas capas no cambia las dimensiones de la imagen porque es un simple mapeo uno a uno de los valores de activación.
- Capas de reducción: se realiza una reducción en el alto y ancho de la imagen pero no en la profundidad.
- Capa tradicional: esta es la última capa y finaliza con la cantidad de neuronas que queremos clasificar.

3.2. Capas convolucionales

3.2.1. Convolución

La convolución es una operación lineal que describe la superposición de dos funciones al combinarlas.Multiplica los valores de las funciones en todos los puntos de superposición, y suma los productos para crear una nueva función, es decir realiza un producto punto, y este se representa con una nueva función.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau.$$

La función *kernel* o núcleo es función de las variables en los dos espacios y define la transformación integral, como en la transformada de Laplace para que la integral

pueda converger el núcleo es la función $f(t) = 0$ para $t \in (-\infty, 0)$ y $f(t) = e^{-st}$ para $t \in (0, \infty)$.

Dependiendo de su aplicación se pueden sustituir las funciones con señales, matrices u otro tipo de datos. Para las matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

y

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

tenemos $A(i, j) = a_{ij}$, la forma general de la convolución matricial es

$$C(i, j) := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{(m-i),(n-j)} b_{(1+i),(1+j)}$$

es decir, $C = A * B$. Donde $C(i, j)$ es la entrada (i, j) de la matriz resultante.

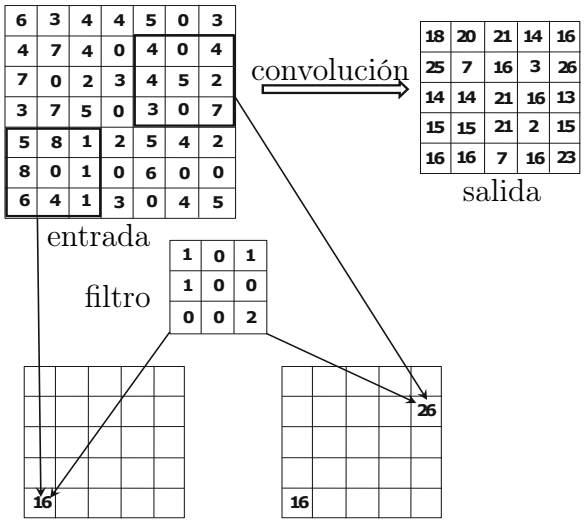
La convolución es el proceso que se utiliza para extraer características o patrones de una matriz de entrada. Para esto se utiliza una matriz de convolución, el *kernel* o **núcleo** que consiste en valores numéricos aleatorios, estos determinan como se transforman los datos de entrada mediante la convolución.

El núcleo se aplica a un área de la matriz de entrada y se calcula un producto escalar entre los valores de entrada y los valores del núcleo. Este producto escalar luego se introduce en una matriz de salida. Luego, el núcleo se desplaza paso a paso, repitiendo el proceso hasta que el núcleo haya recorrido toda la matriz de entrada. La expresión general es

$$G(x, y) = \Phi(i, j) * F(x, y) = \sum_{i=a}^{a} \sum_{j=-b}^{b} \Phi(i, j) F(x - i, y - j).$$

Donde $F(x, y)$ es la matriz de entrada es decir la imagen original, $\Phi(i, j)$ es el núcleo o filtro que se le aplica a la imagen y $G(x, y)$ es la matriz de salida o la imagen resultante. Cada elemento del filtro se considera en los intervalos $-a \leq i \leq a$

y $-b \leq j \leq b$. Ver Figura 3.1.



$$(5 \times 1) + (8 \times 0) + (1 \times 1) + (8 \times 1) + (0 \times 0) + (1 \times 0) + (6 \times 0) + (4 \times 0) + (1 \times 2) = 16$$
$$(4 \times 1) + (0 \times 0) + (4 \times 1) + (4 \times 1) + (5 \times 0) + (2 \times 0) + (3 \times 0) + (0 \times 0) + (7 \times 2) = 26$$

Figura 3.1. Un ejemplo de convolución entre una matriz de 7×7 y un núcleo de 3×3.

- *Padding* o **relleno** se suele utilizar cuando los filtros no se ajustan a la imagen de entrada. Esto establece todos los elementos que quedan fuera de la matriz de entrada en cero, lo que produce una salida mayor o del mismo tamaño. El *padding* conserva el tamaño de la imagen original. Entonces, si una matriz $n \times n$ convoluciona con una núcleo $k \times k$ con *padding* p, entonces el tamaño de la imagen de salida será

$$(n + 2p - k + 1) \times (n + 2p - k + 1)$$

donde $p = 1$ en este caso. Ver Figura 3.2.

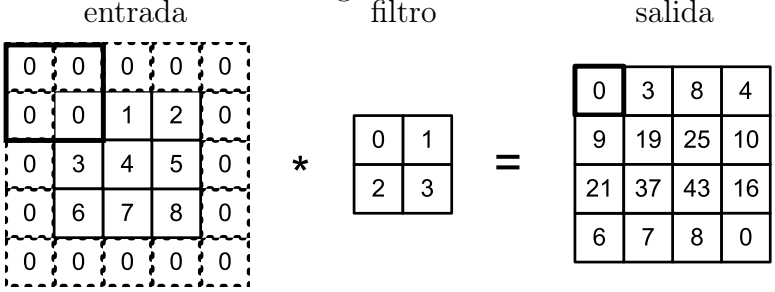


Figura 3.2. Un ejemplo de convolución con padding.

- *Stride* o **zancada** es la distancia, o número de píxeles, que el filtro se mueve sobre la matriz de entrada. Para el *padding* p, el tamaño del filtro $k \times k$ y el

tamaño de la imagen de entrada $n \times n$ y el *stride* s nuestra dimensión de la imagen de salida será

$$\left[\frac{n+2p-k+1}{s}+1\right] \times \left[\frac{n+2p-k+1}{s}+1\right].$$

Ver la Figura .

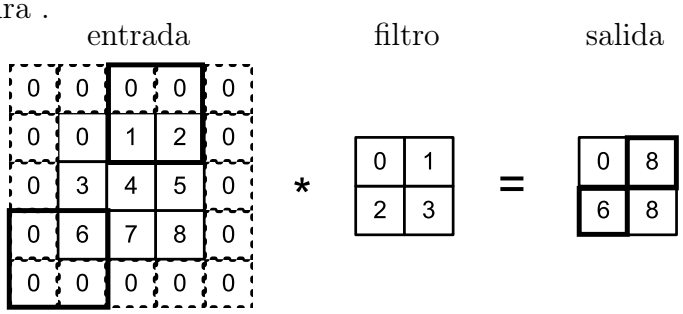


Figura 3.3. Un ejemplo de convolución con stride de 3 y 2 para altura y ancho, respectivamente.

3.2.2. Convolución en el procesamiento de imágenes

La fase de extracción de características se asemeja al proceso que estimula las células de la corteza visual. Esta fase se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo siendo estas activadas por características cada vez más complejas.

Hay varias características que una red neuronal convolucional aprenderá durante el entrenamiento y poco a poco aprenderá características más complejas, una de las más importantes son los bordes de un objeto en la imagen de entrada y se extraen características como estas realizando convoluciones entre la matriz inicial y los filtros correspondientes. Básicamente, los bordes son cambios repentinos de discontinuidades en una imagen.

Las transiciones significativas en una imagen se denominan bordes. La detección de bordes es la primera fase del reconocimiento de imágenes. El borde es útil porque marca los límites y las divisiones del plano, objeto o apariencia de otros lugares. Hay dos enfoques para la detección de bordes, uno basado en gradientes y el segundo basado en Laplaciano [1].

El gradiente utiliza la derivada de primer orden de la imagen. Las derivadas de primer orden son muy sensibles al ruido y producen bordes gruesos. El gradiente de

una imagen en cualquier píxel se define como un vector perpendicular al borde,

$$G\left[f(x,y)\right] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}.$$

Donde el vector G apunta en la dirección de variación máxima de f en el punto (x, y) por unidad de distancia con la magnitud y dirección dadas por

$$|G| = \sqrt{G_x^2 + G_y^2}, \qquad \phi(x,y) = \tan^{-1} \frac{G_y}{G_x}.$$

3.2.2.1. Operador de Sobel

Este operador calcula el gradiente de la intensidad de una imagen en cada píxel. Para cada punto, este operador devuelve la magnitud del mayor cambio posible, la dirección de este y el sentido en escala de grises. Se utilizan dos filtros de 3×3, uno para bordes verticales y otro para horizontales, sea $F_{(i,j)}$ la imagen original, para cada punto se calculan las derivadas de la siguiente manera

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Estos filtros están diseñados para responder al máximo a los bordes que se extienden vertical y horizontalmente en relación con la matriz de píxeles.

3.3. Capas de reducción

La capa de reducción realiza una reducción de dimensionalidad, reduciendo la cantidad de parámetros en la entrada. De manera similar a la capa convolucional, la operación de agrupación barre un filtro por toda la entrada, pero la diferencia es que este filtro no tiene ningún peso. En cambio, el filtro aplica una función de agregación a los valores dentro del campo receptivo, completando la matriz de salida. Hay dos tipos principales de *pooling*:

- *Max pooling:* a medida que el filtro se mueve a través de la entrada, selecciona el píxel con el valor máximo para enviar a la matriz de salida. Este es el tipo de *pooling* más utilizado.

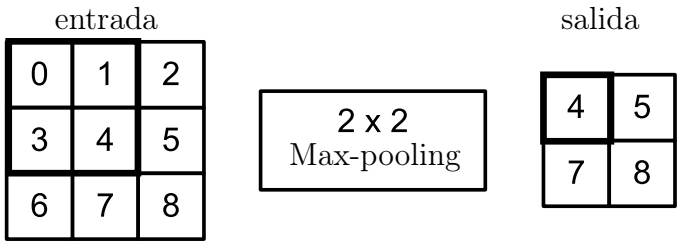


Figura 3.4. Max-pooling con una ventana de pooling de 2×2. máx$(0, 1, 3, 4) = 4$.

- *Average pooling:* a medida que el filtro se mueve a través de la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida.

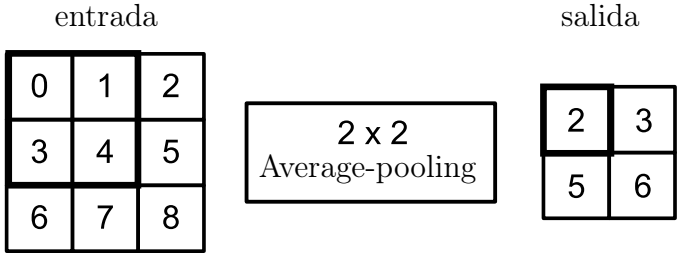


Figura 3.5. Average-pooling con una ventana de pooling de 2×2. avg$(0, 1, 3, 4) = 2$.

Si bien se pierde información en la capa de agrupación, estas ayudan a reducir la complejidad, mejorar la eficiencia y limitar el riesgo de sobreajuste.

3.3.1. Operación de *flatten*

El objetivo de la capa de aplanamiento es convertir los datos en una matriz unidimensional para alimentar la siguiente capa. Aplanamos la salida de la capa convolucional en un único vector de características largo.

Actúa como un puente entre las capas convolucionales y de *pooling*, que extraen características espaciales, y las capas completamente conectadas, que realizan tareas de clasificación o regresión.

3.4. Capa completamente conectada

Las capas completamente conectadas tienen como objetivo sumar los pesos de las capas anteriores con características, indicando la combinación precisa de características para determinar un resultado de salida específico. Después del proceso de capa convolucional y capa de agrupación, la red ya ha extraído con éxito las características de la imagen de entrada. La función de las capas completamente conectadas

es proyectar las características a la capa clasificadora para marcar la etiqueta en la entrada o retroalimentación para mejorar los parámetros.

Esta capa realiza la tarea de clasificación en función de las características extraídas a través de las capas anteriores y sus diferentes filtros. Mientras que las capas convolucionales y de agrupación tienden a usar funciones ReLu, las capas completamente conectadas generalmente aprovechan una función de activación softmax o sigmoidal para clasificar las entradas de manera adecuada, lo que produce una probabilidad entre 0 a 1.

3.5. Entrenamiento

El entrenamiento de una red neuronal convolucional se define como el proceso de ajustar los coeficientes de los filtros. En primer lugar, la CNN inicia con los coeficientes aleatorios. Durante el entrenamiento de CNN, la red neuronal se alimenta con un gran conjunto de datos de imágenes etiquetadas con sus correspondientes etiquetas de clase. La red CNN procesa cada imagen asignando sus valores de forma aleatoria y luego realiza comparaciones con la etiqueta de clase de la imagen de entrada.

Si la salida no coincide con la etiqueta de clase lo que ocurre principalmente inicialmente al comienzo del proceso de entrenamiento y, por lo tanto, realiza un pequeño ajuste respectivo en los pesos de sus neuronas CNN para que la salida coincida correctamente con la imagen de la etiqueta de clase.

Las correcciones al valor de los coeficientes se realizan mediante una técnica conocida como retropropagación. La retropropagación optimiza el proceso de ajuste y facilita los ajustes para una mayor precisión. Cada ejecución del entrenamiento del conjunto de datos de imágenes se denomina época o *epoch*.

La retropropagación mediante convoluciones es similar a la retropropagación con transformaciones lineales en una red neuronal hacia adelante. Así como la retropropagación en redes neuronales hacia adelante, desde la capa $(i+1)$ a la capa i se logra multiplicando las derivadas del error con respecto a la capa $(i+1)$ con la transpuesta de la matriz de propagación directa entre las capas i y $(i+1)$.

En las capas con función de activación ReLU la retropropagación funciona como una red neuronal tradicional. Para las capas de *max pooling* solo es necesario identificar qué unidad es el valor máximo en un grupo, la derivada parcial de la pérdida con respecto al estado agrupado regresa a la unidad con valor máximo.

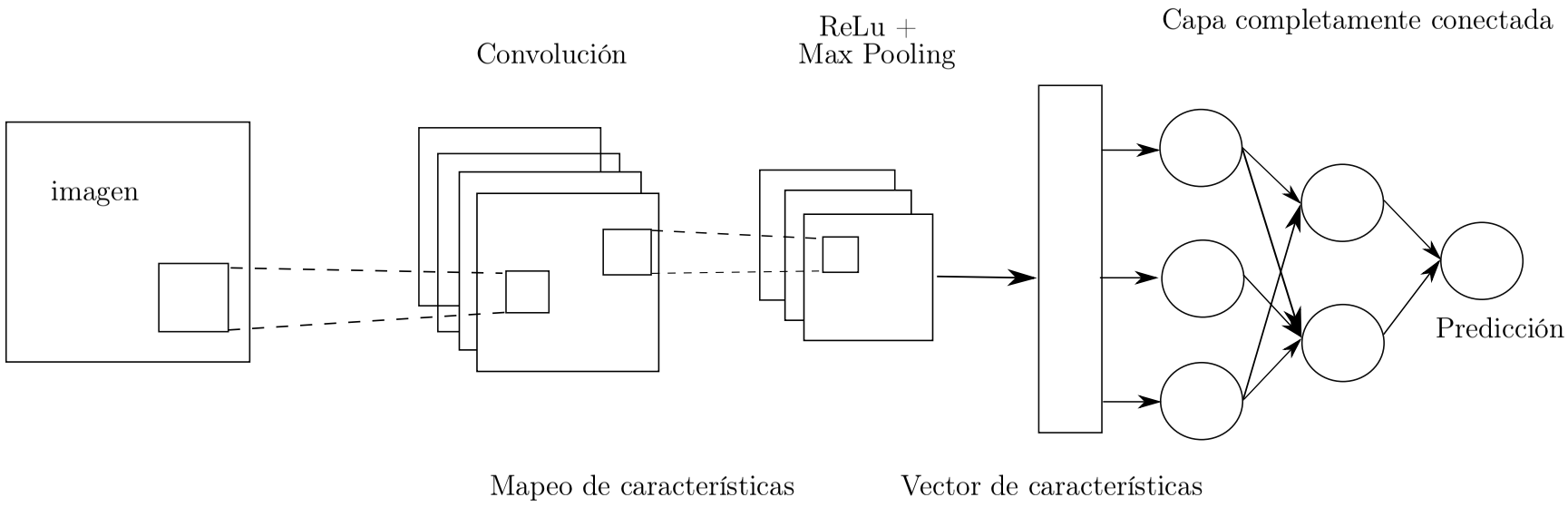


Figura 3.6. Arquitectura básica de una red neuronal convolucional.

A todas las entradas distintas de la entrada máxima en la cuadrícula se les asignará un valor de 0. La CNN pasa por varias series de épocas durante el proceso de entrenamiento, ajustando sus pesos según las pequeñas cantidades requeridas.

Después de cada paso de época, la red neuronal se vuelve un poco más precisa a la hora de clasificar y predecir correctamente la clase de las imágenes de entrenamiento. A medida que la CNN mejora, los ajustes que se realizan a las ponderaciones se vuelven cada vez más pequeños en consecuencia.

3.5.1. Hiperparámetros

1. Learning rate: La tasa de aprendizaje determina el tamaño del paso con el que la red actualiza sus parámetros durante el entrenamiento. Una tasa de aprendizaje elevada puede conducir a una convergencia rápida, pero puede dar lugar a un entrenamiento inestable y oscilante. Una tasa de aprendizaje pequeña puede garantizar un entrenamiento estable y fluido, pero puede dar como resultado una convergencia más lenta. Por lo tanto, es importante experimentar con diferentes ritmos de aprendizaje y elegir el que ofrezca el mejor equilibrio entre velocidad de entrenamiento y estabilidad.

2. Batch: El tamaño del lote determina la cantidad de muestras que procesa la red en cada iteración de entrenamiento. Un tamaño de lote mayor puede reducir la varianza de las estimaciones del gradiente y mejorar la estabilidad del entrenamiento. Sin embargo, también aumenta los requisitos de memoria y puede provocar una convergencia más lenta. Un tamaño de lote más pequeño puede reducir los requisitos de memoria y mejorar la velocidad de convergencia, pero puede generar estimaciones de gradiente ruidosas. Por lo tanto, es importante experimentar con diferentes tamaños de lotes y elegir el que ofrezca el mejor equilibrio entre estabilidad y velocidad.

3. Epoch: El número de épocas es el número de veces que todo el conjunto de entrenamiento pasa por la red neuronal. Se aumenta el número de épocas hasta que haya una pequeña brecha entre el error de prueba y el error de entrenamiento.

4. Número de capas y unidades ocultas: Generalmente es bueno agregar más capas hasta que el error de prueba ya no mejore. La desventaja es que resulta computacionalmente costoso entrenar la red. Tener una pequeña cantidad de

unidades puede provocar un desajuste, mientras que tener más unidades no suele ser perjudicial con una regularización adecuada.

5. Inicialización de pesos: Se debe inicializar los pesos con números aleatorios pequeños para evitar neuronas muertas, pero no demasiado pequeños para evitar un gradiente cero. La distribución uniforme suele funcionar bien.

3.6. Evaluación

En la evaluación de modelos de clasificación el concepto básico es la noción de fallo. Si la aplicación de los modelos de clasificación en un caso seleccionado conduce a la predicción de una clase que es diferente de los ejemplos de clase reales, entonces hay un error en la clasificación. Si algún error es igualmente importante, entonces el número total de errores en el conjunto observado puede ser un indicador del trabajo del clasificador. Podemos clasificar instancias producidas por el modelo en los datos de prueba en:

- Verdaderos positivos (TP): ocurren cuando el modelo predice con precisión un punto de datos positivo.

- Verdaderos negativos (TN): ocurren cuando el modelo predice con precisión un punto de datos negativo.

- Falsos positivos (FP): ocurren cuando el modelo predice incorrectamente un punto de datos positivo.

- Falsos negativos (FN): ocurren cuando el modelo predice erróneamente un punto de datos negativo.

Hay muchas formas de medir el rendimiento de la clasificación, las más utilizadas son:

- Precisión: o *accuracy*, simplemente mide la frecuencia con la que el clasificador predice correctamente. Podemos definir la precisión como la relación entre el número de predicciones correctas y el número total de predicciones.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

- Sensibilidad: o *recall* Explica cuántos de los casos positivos reales pudimos predecir correctamente con nuestro modelo. Es una métrica útil en los casos en los que el falso negativo es más preocupante que el falso positivo.

$$\text{Recall} = \frac{TP}{TP + FN}.$$

- Matriz de confusión: es una medida de rendimiento para los problemas de clasificación del aprendizaje automático donde el resultado puede ser dos o más clases. Es una tabla con combinaciones de valores previstos y reales.

  Las predicciones se resaltan y dividen por clase (verdadero/falso), antes de compararlas con los valores reales. El tamaño de la matriz es compatible con la cantidad de clases en la columna de etiqueta. En una clasificación binaria, la matriz será 2×2. Si hay 3 clases, la matriz será 3×3, y así sucesivamente. Básicamente, esta matriz permite a determinar si el modelo de clasificación está optimizado. Muestra qué errores se están cometiendo y ayuda a determinar su tipo exacto.

- AUC-ROC: Una curva ROC (curva de característica operativa del receptor) es una curva de probabilidad que traza la TPR (tasa de verdaderos positivos) frente a la FPR (tasa de falsos positivos) en varios valores umbral y separa la señal del ruido.

  El área bajo la curva (AUC) es la medida de la capacidad de un clasificador para distinguir entre clases.

3.7. Aplicaciones

- Detección de objetos: CNN puede detectar y localizar objetos en imágenes o videos.

- Segmentación de imágenes: las CNN pueden segmentar imágenes en diferentes regiones y etiquetar cada región con una clase semántica.

- Crear imágenes: las CNN pueden crear nuevas imágenes o manipular las existentes.

- Análisis de vídeo: las CNN se pueden utilizar para la detección de acciones, el seguimiento de objetos y la segmentación de escenas de vídeo.

- Procesamiento del lenguaje natural: las CNN se pueden utilizar para tareas de clasificación de texto, análisis de sentimientos y traducción de idiomas.

4. PLANTEAMIENTO DEL PROBLEMA Y METODOLOGÍA

4.1. Planteamiento del problema

Debido a la pandemia la educación migró al entorno virtual lo que provocó el alto volumen de asignaciones en formato de hojas escaneadas o fotografías y dificulta el proceso de calificación. Para esto, se buscó implementar una herramienta basada en redes neuronales que identifique las respuestas en tareas hechas a mano.

4.2. Propuesta

Se propone un sistema de calificación de tareas con respuestas numéricas escritas a mano en un formato establecido, a través de una red neuronal convolucional que reconozca los números de las respuestas y luego evalúe los resultados para calificar la tarea. Este sistema tendrá la siguiente estructura:

1. Proceso de obtención de las imágenes individuales de cada número de las hojas de respuestas escaneadas.
2. Clasificación de las imágenes a través de la red neuronal convolucional.
3. Evaluación de las respuestas obtenidas por las imágenes clasificadas por la red.

4.2.1. Requerimientos

El sistema se implementó en cuadernos de Google Colab que ejecuta el código en los servidores de la nube de Google utilizando únicamente un navegador. Los cuadernos de Colab se basan en los cuadernos de Jupyter y son muy parecidos en su estructura y funcionamiento. Tienen integrado Python 3 con sus librerías principales y la posibilidad de activar Unidad de Procesamiento Gráfico (GPU) de manera gratuita durante 12 horas, en las versiones de paga se pueden acceder a más

GPU por más tiempo. Para este proyecto fue necesario utilizar la versión gratuita con una cuenta de gmail específicamente para este proceso. La versión de Python utilizada fue la 3.6 y las librerías y sus versiones utilizadas son:

- pdf2image 1.16.0
- os 3.6
- cv2 4.5.4
- matplotlib 3.4.0
- shutil 1.1
- random 3.6
- numpy 1.21
- tensorflow 2.6
- pandas 1.3.2

4.2.2. Datos

Los datos para probar el sistema de calificación y entrenar la red neuronal fueron obtenidos por parte de los estudiantes del curso de Cálculo 1 y 2 en modalidad virtual de la Escuela de Ciencias Físicas y Matemáticas que se impartía durante cuatro periodos a la semana durante el segundo semestre de 2021. Todos los documentos fueron enviados en la plataforma de Moodle en su versión 4.25 `https://ecfm.usac.edu.gt/portal/` en formato pdf.

- Datos de entrenamiento:

  Para el entrenamiento de la red neuronal convolucional, se les solicitó a los estudiantes realizar planas de los dígitos del 0 al 9 para poder entrenar a la red con los mismos trazos con los que se iba a clasificar más adelante. Se les proporcionó un formato el cual tenían que imprimir y realizar las planas con lapicero negro y luego escanearlas y subirlas a la plataforma de Moodle. Ver Figura 4.1.

  Luego de esto, los documentos eran descargados y almacenados en una carpeta en Google Drive de la cuenta del proyecto. En un cuaderno de Colab se

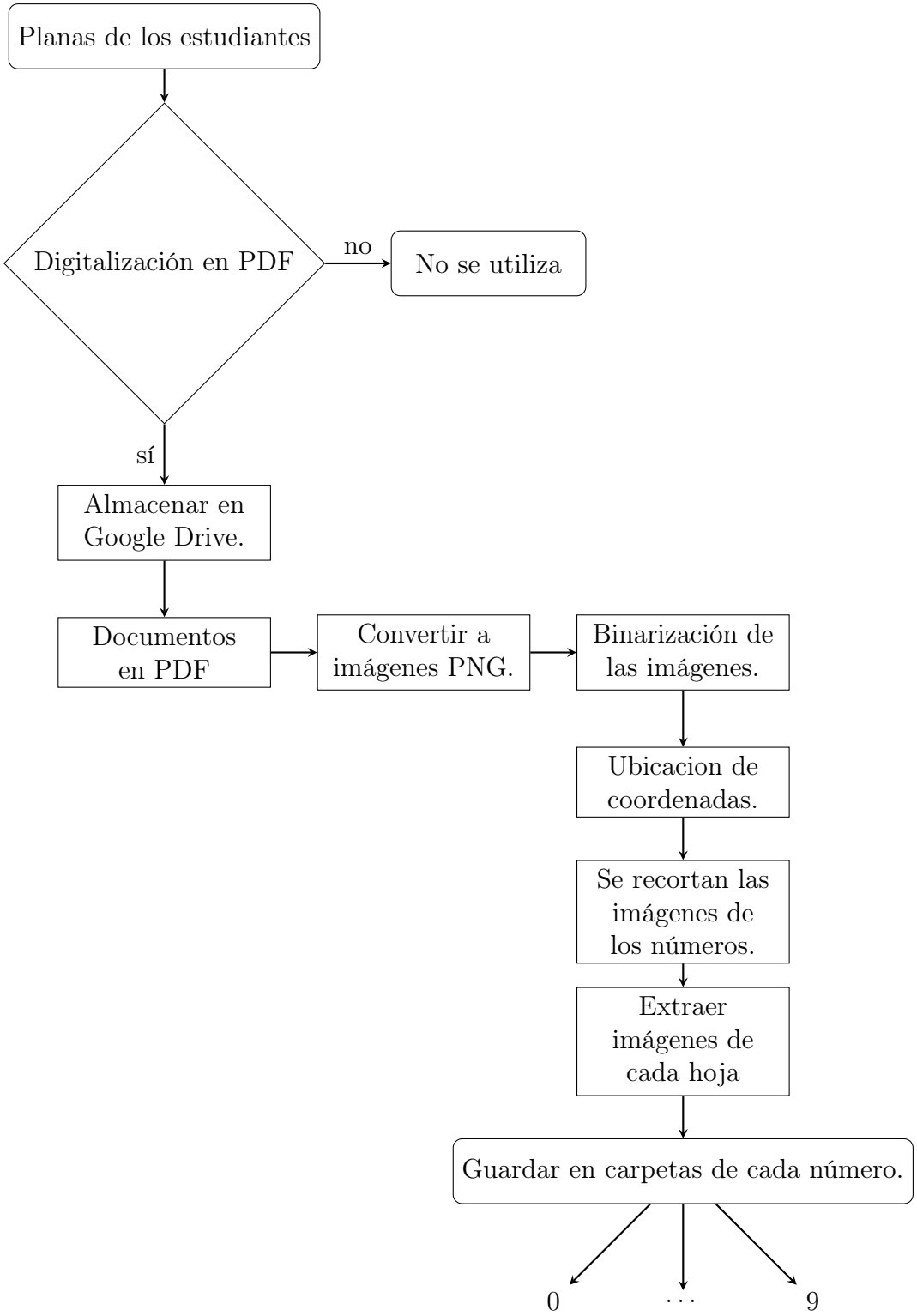


Figura 4.1. Proceso de construcción del conjunto de entrenamiento. Fuente: elaboración propia

cargaron los documentos y se convirtieron los documentos a imágenes png, esto para poder transformarlas con una binarización[1] en blanco y negro y estandarizándolas en tamaño.

Con el formato dado, se tiene la ubicación del primer recuadro de las planas y de esta manera se extrae las coordenadas de la primera imagen para extraer. Luego se itera en cada uno de los números para obtener el cambio en x y cambio en y para obtener todos los recuadros de la hoja, se guardan las coordenadas de cada recuadro y así extraer cada una de las imágenes con cada dígito. Por último se guardan todas las imágenes extraídas en subcarpetas etiquetadas para cada dígito en la carpeta *Full DataSet* en Google Drive. Se obtuvo un total de 32 documentos escaneados, de los cuales se utilizaron 20 para el entrenamiento y 12 para la fase de validación, obteniendo un total de 1950 imágenes.

- Datos de producción: Para probar el sistema de calificación, se les asignó a los estudiantes 3 problemas con solución numérica, la cual pudieran colocar en una hoja de respuestas dada previamente donde tenían que escribir cada dígito de la respuesta en un recuadro, cada respuesta tenía un máximo de tres dígitos.

4.2.3. Sistema de calificación

Para el sistema de calificación se utilizó también la plataforma de Moodle, se colocó en la asignación tres problemas a resolver con respuestas numéricas. En la asignación se restringió los documentos que se podían recibir a solo archivos en formato PDF. Los estudiantes escaneaban la hoja de respuestas y la subían a la plataforma, luego de la fecha de entrega se descargaban los documentos recibidos y se guardaban en Google Drive en la carpeta asignada al número de tarea correspondiente. Ver Figura 4.2.

El proceso de extracción, clasificación y calificación está construido en un cuaderno de Google Colab. Se conecta a Google Drive para extraer cada uno de los archivos, con una función se buscan los bordes de los recuadros de las respuestas, se extraen las imágenes de cada dígito y se clasifican con la red neuronal, una vez clasificadas se guardan las predicciones en un diccionario de Python y se comparan

[1]La binarización es una técnica de procesamiento de imágenes que se utiliza para convertir una imagen en escala de grises. La imagen creada como resultado de la binarización contiene solo dos valores de píxel, generalmente 0 y 1.

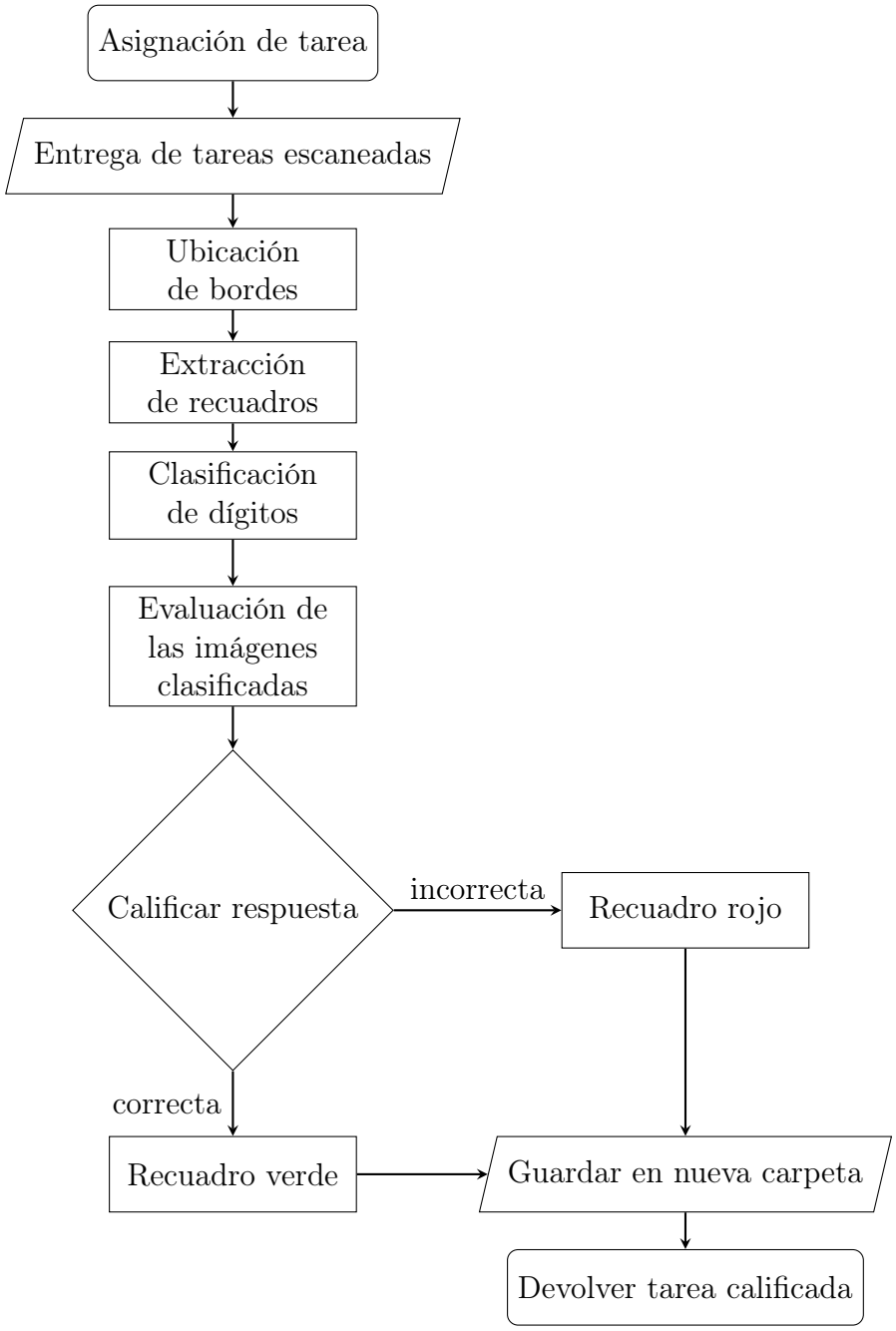


Figura 4.2. Flujo del sistema de calificación. Fuente: elaboración propia.

con las respuestas correctas también definidas en un diccionario de Python llamado «clave».

Al comparar cada llave del diccionario de predicciones con la del diccionario clave se crea un nuevo diccionario para almacenar la evaluación, «correcto» si son igual e «incorrecto» si no lo son. Por último, con las coordenadas de los recuadros guardadas se dibuja un recuadro encima de cada dígito, verde para correcto, rojo para incorrecto, y se guarda el nuevo documento en una nueva carpeta correspondiente para cada tarea. Ver Figuras 4.3 y 4.4.

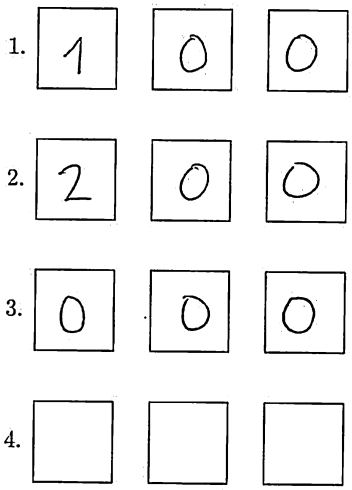


Figura 4.3. Ejemplo de una de las tareas entregadas por los estudiantes.

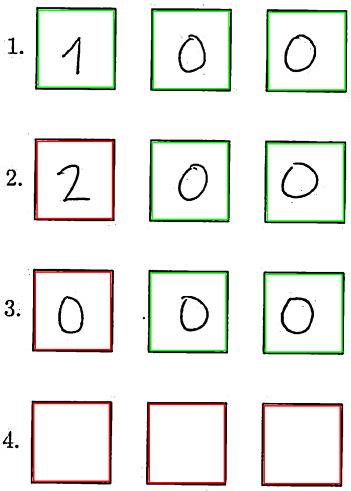


Figura 4.4. Ejemplo del documento devuelto por el sistema de calificación.

4.2.4. Modelo de CNN de clasificación

Para este trabajo se utilizaron imágenes de 128×128 píxeles, extraídas de las planas y las hojas de respuestas. La estructura propuesta del modelo de clasificación se describe en la tabla, está basada en la estructura descrita en un artículo para clasificación de hojas de plantas [6] y consta de 10 capas que incluyen los elementos descritos en la Tabla 4.1. Para una visualización esquemática ver la Figura 4.5.

Tabla 4.1. Arquitectura propuesta para la CNN (imágenes de entrada 128×128). Fuente: Elaboración propia.

Tipo	Función activación	Tamaño filtro	Filtros	Stride	Tamaño salida
Conv1	ReLu	8×8	400	2	61×61
MaxPool1		7×7	400	2	28×28
Conv2	ReLu	5×5	100	2	12×12
MaxPool2		6×6	100	2	4×4
Conv3	ReLu	2×2	50		3×3
Conv4	ReLu	3×3	50		1×1
Flatten					50
Dense1	ReLu				50
Dense2	ReLu				25
Dense3	SoftMax				10

Para las capas convolucionales y completamente conectadas se utilizó la función de activación de ReLu, ver página 15, ya que por trabajar con pixeles de imágenes es necesario que los valores se mantengan positivos. Para la capa de salida se utilizó la función SoftMax ya que es la más adecuada para modelos de clasificación y en este caso para cada neurona de la capa de salida la función devuelve una probabilidad para cada dígito, siendo la más alta la clasificación final.

Como función de error se utilizó el Error de Entropía Cruzada Categórica, esta función se utiliza habitualmente en redes neuronales con activación softmax en la capa de salida para tareas de clasificación de múltiples clases mutuamente excluyentes [8]. Al minimizar la pérdida, el modelo aprende a asignar mayores probabilidades a la clase correcta y, al mismo tiempo, reduce las probabilidades de las clases incorrectas, lo que mejora la precisión.

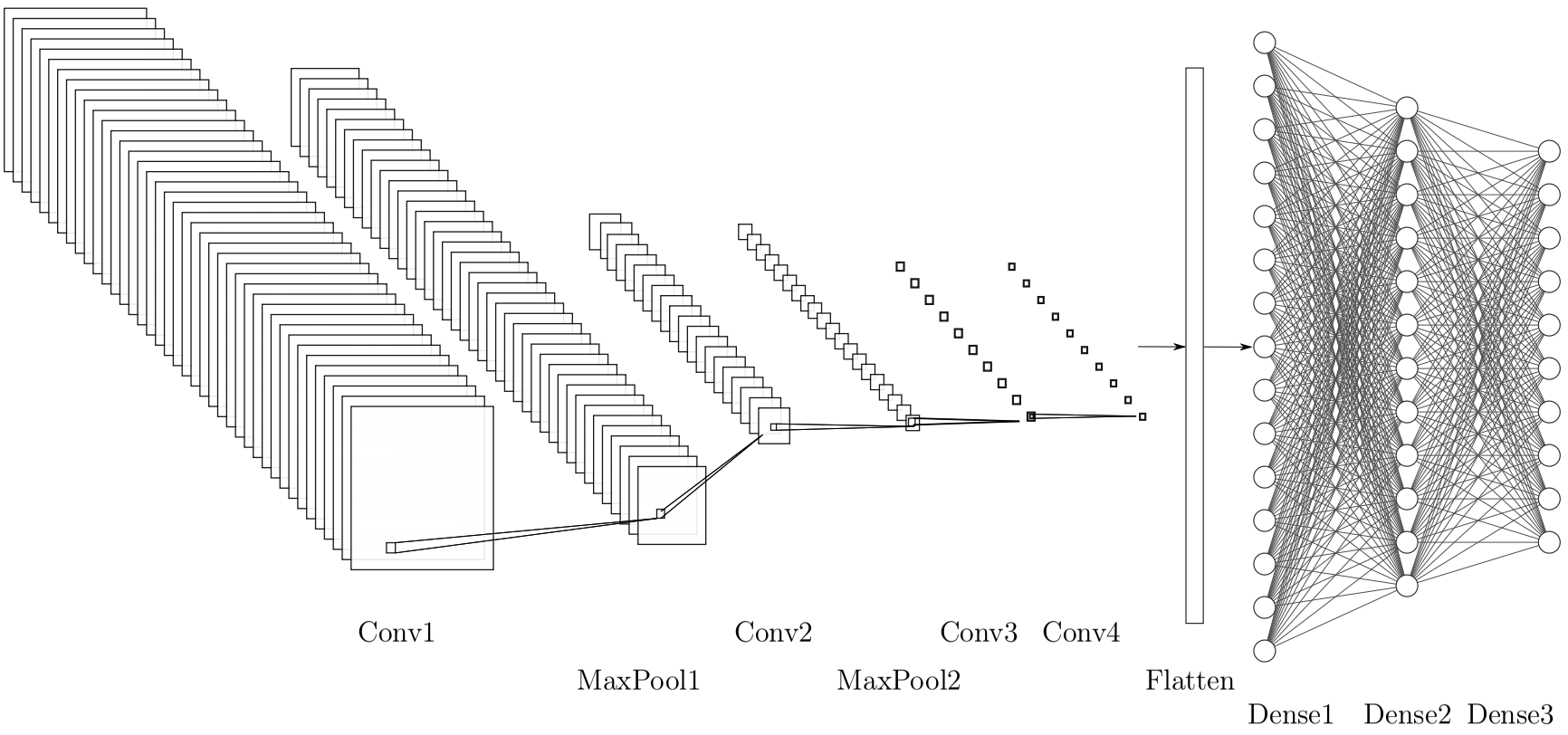


Figura 4.5. Arquitectura de la red neuronal convolucional propuesta. Fuente: Elaboración propia.

4.3. Resultados

4.3.1. Entrenamiento del modelo

Para entrenar el modelo se utilizaron 1250 imágenes previamente ajustadas al tamaño y a las características establecidas para el entrenamiento y 310 para la validación. El entrenamiento consistió en 150 ciclos en las que el conjunto pasó por la red, segmentado en lotes de 20 imágenes distribuidas uniformemente. Tomó un tiempo de 3 horas, 11 minutos y 56 segundos obteniendo una precisión de 93.75 % para el entrenamiento y una precisión de 81.48 % para validación. Ver Figura 4.6

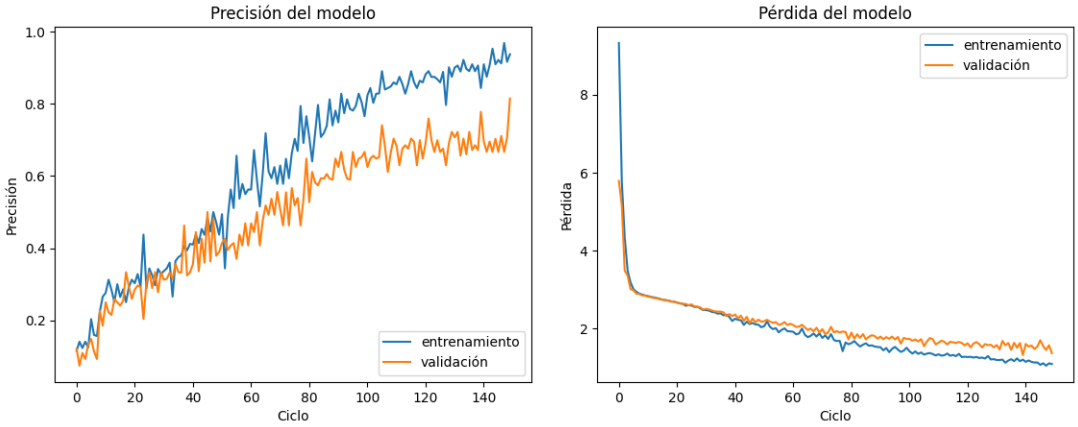


Figura 4.6. Gráficas que muestran la precisión y la pérdida del modelo para el conjunto de entrenamiento y el conjunto de validación. Fuente: Resultados obtenidos en Google Colab.

4.3.1.1. Comparación de conjuntos de entrenamiento

Para probar la eficiencia de la arquitectura del modelo se realizó un entrenamiento con un subconjunto de la base de datos MNIST. Se utilizaron 6000 imágenes para el entrenamiento y 1000 para la validación, todas con las mismas características de las imágenes del conjunto de entrenamiento. Para este conjunto se obtuvieron los resultados que se muestran en la Figura 4.7.

Con un tiempo de entrenamiento de 3 minutos y 7 segundos.

4.3.2. Evaluación del modelo

Para evaluar el modelo se utilizaron dos conjuntos de imágenes, el primero de prueba de 390 imágenes obtenidas de las imágenes de planas escaneadas y el segundo

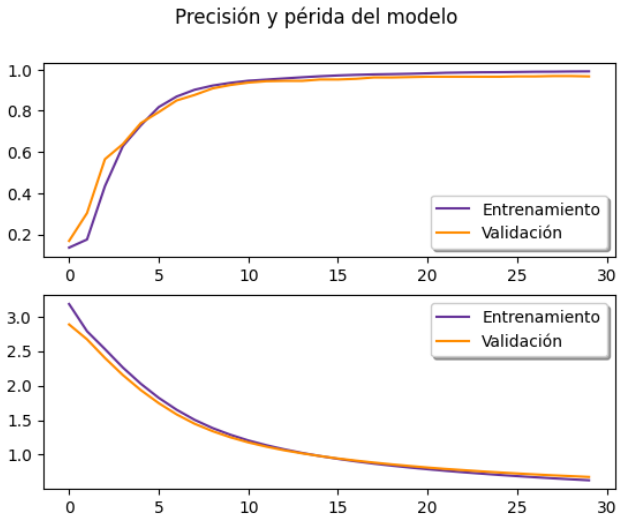


Figura 4.7. Gráficas que muestran la precisión y la pérdida del modelo entrenado con el subconjunto de la base de datos MNIST. Fuente: Resultados obtenidos en Google Colab.

de 779 imágenes que no fueron digitalizadas con escáner, es decir ninguna de esas imágenes estaba dentro del entrenamiento del modelo.

Con ambos se obtuvo un la curva ROC del modelo de predicción. Por ser un modelo de clasificación multicategoría se obtiene una curva por cada clase es decir:

- clase del 0 vs clases del 1-9
- clase del 1 vs clases del 0,2-9
- etc

Para el conjunto de prueba se obtuvo un AUC de 0.9862 y un ROC AUC score de 0.9874, ver Figura 4.8.

Para el conjunto de reserva se obtuvo un AUC de 0.9632 y un ROC AUC score de 0.9641, ver Figura 4.9.

También con el conjunto de prueba se obtuvo una precisión del 84.37 %, con sensibilidad de 83.24 % y la matriz de confusión dada en la Figura 4.10.

Se realizó también la clasificación del conjunto de reserva, con el modelo entrenado con planas de los estudiantes y con el entrenado con base de datos MNIST. Los resultados de la clasificación están dados en la Figura 4.11.

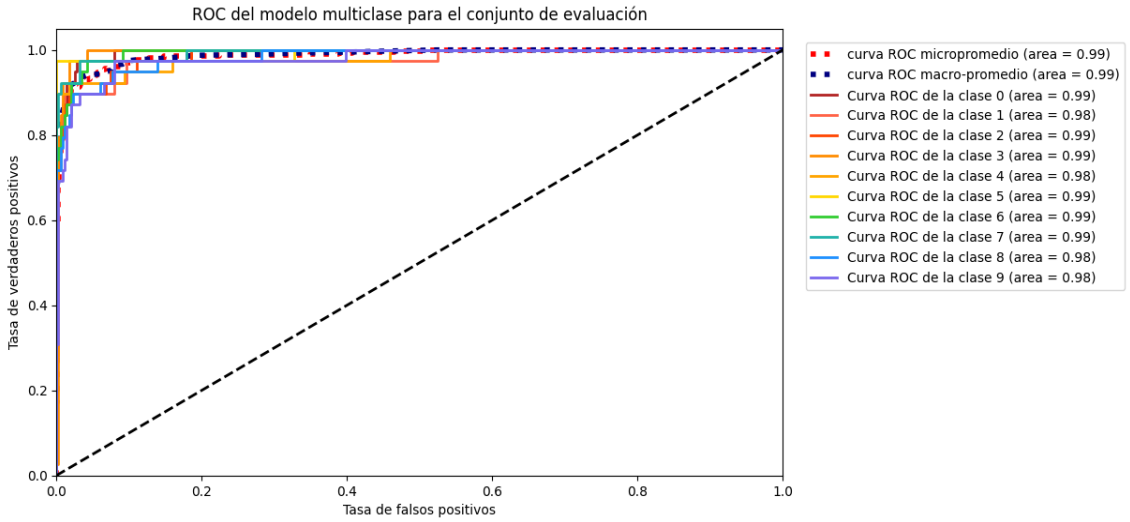


Figura 4.8. Curva ROC del modelo multiclase para el conjunto de prueba. Fuente: Resultados obtenidos en Google Colab.

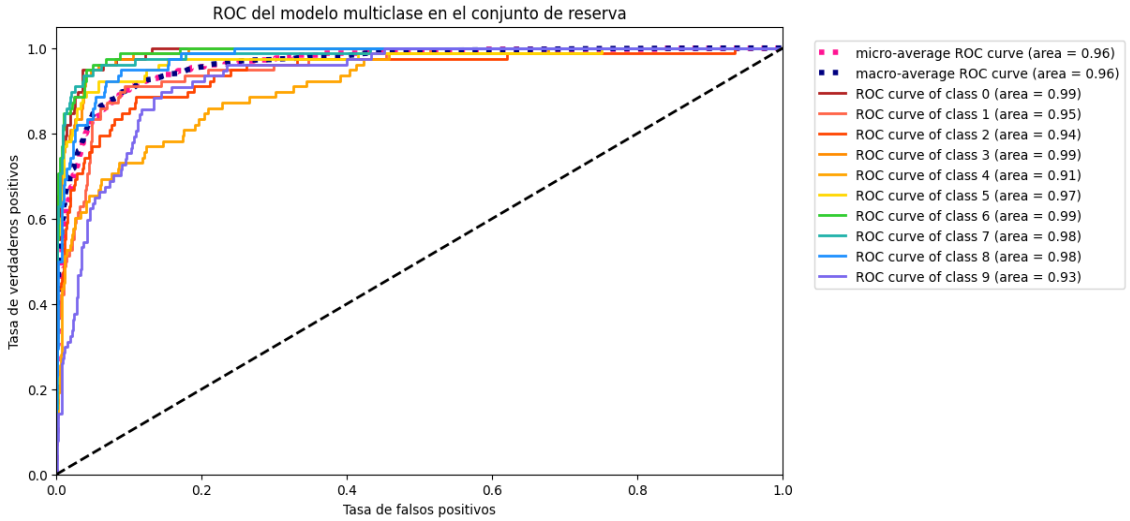


Figura 4.9. Curva ROC del modelo multiclase para el conjunto de reserva. Fuente: Resultados obtenidos en Google Colab.

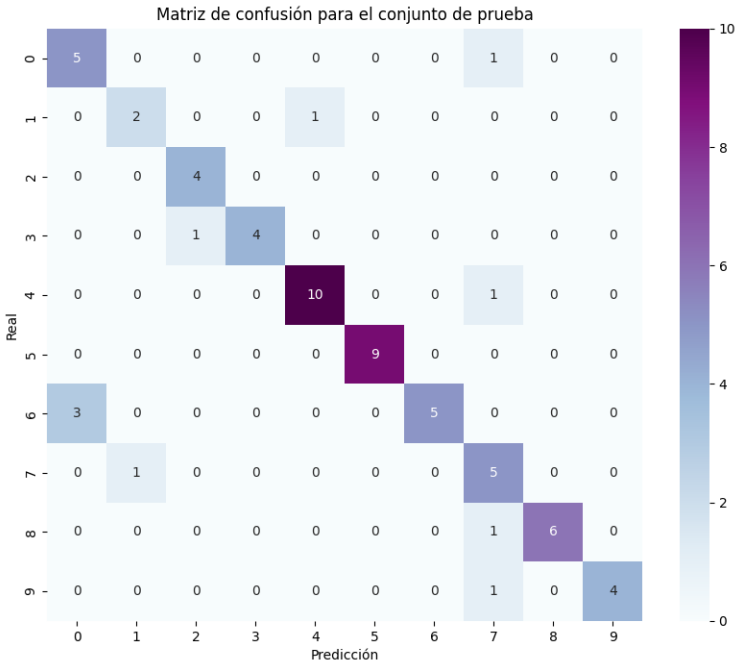


Figura 4.10. Matriz de confusión para del modelo con el conjunto de prueba. Fuente: Resultados obtenidos en Google Colab.

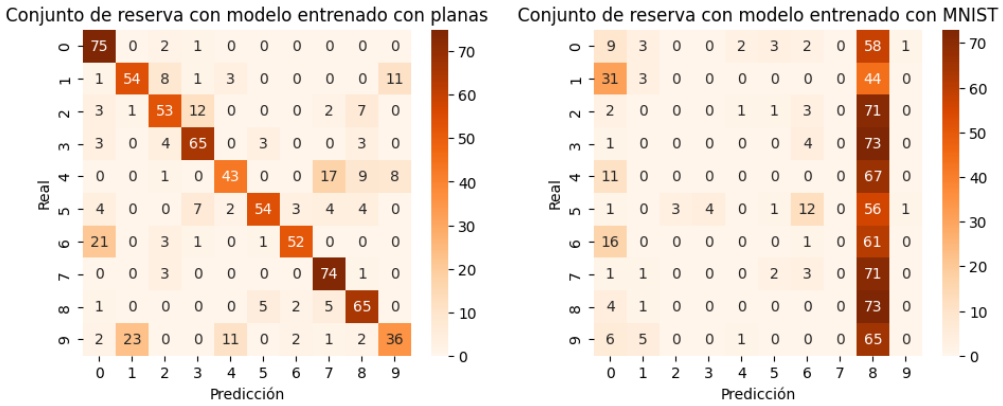


Figura 4.11. Matrices de confusión para la clasificación del conjunto de prueba con el modelo entrenado por dos conjuntos diferentes. Fuente: Resultados obtenidos en Google Colab.

4.3.3. Tareas calificadas

Se asignaron tres tareas en la plataforma Moodle, en la asignación estaba el formato a utilizar, los estudiantes tendrían que descargarlo, imprimirlo, escribir sus respuestas y digitalizar el documento en un scanner para subirlo en la asignación. Estas tareas fueron calificadas con el sistema de calificación y por el catedrático del curso para comparar la eficiencia del sistema. Para cada tarea se obtuvieron los siguientes resultados:

Tabla 4.2. Resultados del sistema de calificación con las tareas. Fuente: Elaboración propia.

Tarea	Tareas enviadas	Tareas calificadas	Calificadas correctamente	Eficiencia
Tarea 1	21	17	13	76 %
Tarea 2	21	16	12	75 %
Tarea 3	21	16	14	88 %

Para cada tarea, los resultados de clasificación de las imágenes por parte del modelo fueron los siguientes:

- Tarea 1

Tabla 4.3. Matriz de confusión para la tarea 1. Fuente: Resultados obtenidos en Google Colab.

Real/ Predicho	0	1	2	3	4	5	6	7	9
0	112	9	0	1	0	0	0	6	0
1	0	30	0	0	0	0	0	0	0
2	1	0	2	0	0	0	0	0	0
4	0	3	0	0	8	0	1	3	1
5	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0

Precisión: 86.03 % Sensibilidad: 84.02 %.

- Tarea 2

Tabla 4.4. Matriz de confusión para la tarea 2. Fuente: Resultados obtenidos en Google Colab.

Real/ Predicho	0	1	2	3	4	5	6	7
0	51	1	0	0	2	0	0	3
1	0	36	1	0	0	0	0	1
2	1	0	40	0	0	0	0	1
3	4	0	3	4	0	0	0	0
4	0	1	0	0	6	0	0	2
5	6	2	0	0	0	15	0	0
6	1	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	21
9	0	1	0	0	0	0	0	0

Precisión: 85.29 % Sensibilidad: 66.41 %.

- Tarea 3

Tabla 4.5. Matriz de confusión para la tarea 3. Fuente: Resultados obtenidos en Google Colab

Real/ Predicho	0	1	2	4	5	6	7
0	111	2	3	1	0	3	4
1	0	16	0	0	0	0	0
2	0	0	19	0	0	0	1
5	9	0	0	1	10	0	1
8	0	0	0	0	0	0	2

Precisión: 85.24 % Sensibilidad: 66.43 %.

4.4. Trabajo Futuro

A partir de este sistema de calificación se puede producir una nueva versión, donde se utilice una herramienta de **Reconocimiento Óptico de Caracteres** (*OCR* por sus siglas en inglés), para reconocer y extraer las imágenes de dígitos sin la necesidad de utilizar un formato de hoja de respuestas o que sean digitalizadas por medio de scanner. Esto permitirá aumentar la cantidad de tareas que se puedan calificar ya que pueden ser digitalizadas por medio de aplicaciones móviles y también podrán utilizarse otros formatos para escribir las respuestas.

CONCLUSIONES

1. La estructura del modelo de clasificación es funcional para la clasificación de imágenes de dígitos con tamaño de 128×128 píxeles ya que el entrenamiento con el conjunto de planas dio una precisión de 0.93 y con el conjunto MNIST dio una precisión de 0.99.

2. Al clasificar el conjunto de reserva con ambos modelos, se puede observar que a pesar de tener una precisión del 99 % en el entrenamiento, el modelo entrenado con el conjunto MNIST obtuvo una precisión del 11.11 % al predecir el conjunto de reserva, esto se debe a que el modelo se sobreajustó al conjunto de MNIST y no es apto para clasificar otro tipo de imágenes, por lo que la mejor opción es entrenar el modelo con las imágenes de las planas hechas por los alumnos.

3. El sistema de calificación muestra una eficiencia entre el 75 % - 88 % para las tres tareas y una precisión entre el 85 % - 86 % del total de imágenes clasificadas para cada tarea. Las tareas que el sistema no logró calificar presentan una calidad baja en la digitalización o un tamaño distinto al formato establecido, por que el error en la clasificación y califcación fue dado por la extracción y transformación de las imágenes.

RECOMENDACIONES

1. Utilizar una herramienta de Reconocimiento Óptico de Caracteres, *OCR* para la extracción de las imágenes en las tareas, de esta manera se podrán recibir documentos digitalizados por otras herramientas y mejorar la identificación y la calidad de las imágenes a ser clasificadas.

2. Seguir alimentando el conjunto de entrenamiento de la red con las planas de futuros estudiantes para mejorar la capacidad de generalización del modelo y evitar el sobreajuste.

3. Para utilizar el sistema de calificación y correr el modelo es necesario utilizar Unidades de procesamiento gráfico *GPU*, por lo que es recomendable utilizar Google Colab ya que es un servicio alojado de Jupyter Notebook y ofrece de manera básica este tipo de recursos de sin algún coste.

BIBLIOGRAFÍA

[1] Aggarwal, C.C.: *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018. ISBN 9783319944630.
https://books.google.com.gt/books?id=achqDwAAQBAJ

[2] Aggarwal, Charu C.: *Linear Algebra and Optimization for Machine Learning: A Textbook*. Springer International Publishing, 2020. ISBN 9783030403430.
https://books.google.com.gt/books?id=k3YjzQEACAAJ

[3] Bondy, J.A. y Murty, U.S.R.: *Graph Theory with Applications*. American Elsevier Publishing Company, 1976. ISBN 9780444194510.

[4] Grimaldi, R.P.: *Discrete and Combinatorial Mathematics*. Pearson Education, 5ª edición, 2006. ISBN 9788177584240.

[5] Hubel, David H y Wiesel, Torsten N: «Receptive fields of single neurones in the cat's striate cortex». *The Journal of physiology*, 1959, **148(3)**, p. 574.

[6] Huynh, Hiep Xuan; Truong, Bao Quoc; Nguyen Thanh, Kiet Tan y Truong, Dinh Quoc: «Plant identification using new architecture convolutional neural networks combine with replacing the red of color channel image by vein morphology leaf». *Vietnam Journal of Computer Science*, 2020, **7(02)**, pp. 197–208.

[7] Kneusel, R.T.: *Math for Deep Learning: What You Need to Know to Understand Neural Networks*. No Starch Press, 2021. ISBN 9781718501904.
https://books.google.com.gt/books?id=rAlNEAAAQBAJ

[8] Torres, J.: *Python Deep Learning: Introducción práctica con Keras y TensorFlow 2*. Alpha Editorial, 2020. ISBN 9789587786415.
https://books.google.com.gt/books?id=0XJ6EAAAQBAJ

A. Construcción del conjunto de entrenamiento

Código para la extracción y transformación de imágenes para los conjuntos de entrenamiento y validación del modelo utilizando las planas enviadas por los estudiantes.

Instalamos librerías no instaladas en Colab.

[ ]:
```
pip install pdf2image
```

```
Collecting pdf2image
  Downloading pdf2image-1.17.0-py3-none-any.whl.metadata (6.2 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from pdf2image) (9.4.0)
Downloading pdf2image-1.17.0-py3-none-any.whl (11 kB)
Installing collected packages: pdf2image
Successfully installed pdf2image-1.17.0
```

[ ]:
```
!apt-get install poppler-utils
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  poppler-utils
0 upgraded, 1 newly installed, 0 to remove and 40 not upgraded.
Need to get 154 kB of archives.
After this operation, 613 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 poppler-utils
amd64 0.62.0-2ubuntu2.12 [154 kB]
Fetched 154 kB in 1s (296 kB/s)
Selecting previously unselected package poppler-utils.
(Reading database ... 148492 files and directories currently installed.)
Preparing to unpack .../poppler-utils_0.62.0-2ubuntu2.12_amd64.deb ...
Unpacking poppler-utils (0.62.0-2ubuntu2.12) ...
Setting up poppler-utils (0.62.0-2ubuntu2.12) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Cargamos las librerias a utilizar, consierando que vamos a obtener archivos de Google drive.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
from os import listdir, mkdir
import cv2
from pdf2image import convert_from_path
import matplotlib.pyplot as plt
import shutil
from random import sample, seed
import numpy as np
```

Definimos rutas generales a utilizar.

[ ]:
```
root = './drive/MyDrive/Entrenamiento/'
path_planas = root + 'Planas/'
```

Cambiamos los .pdf en la ruta de las planas a .png

[ ]:
```
alumnos = [str(x) for x in range(1,21)]
for n in alumnos:
  path = path_planas + 'entrenar_' + n +'.pdf'
  pages = convert_from_path(path, 500)
  for page in pages:
    page.save(path_planas + 'entrenamiento_' + n + '.png', 'PNG')
```

Creamos una carpeta por número para guardar todas las imágenes.

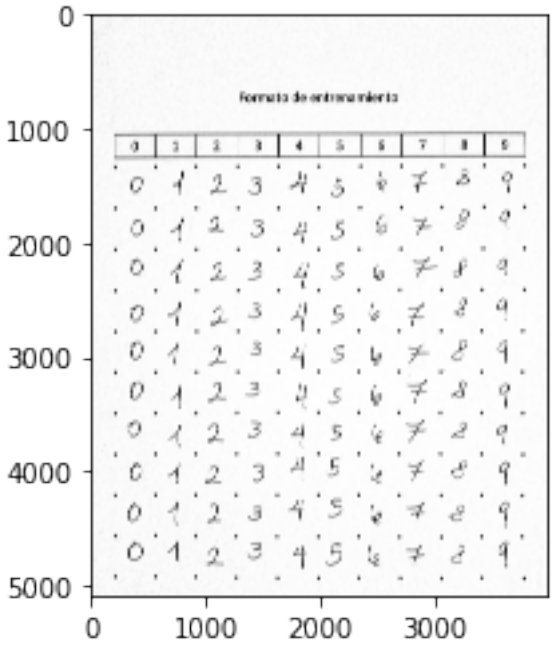
[ ]:
```
path1 = root + 'Full_DataSet/'
mkdir(path1)
for i in range(0,10):
  new_dir = path1 + '/'  + str(i)
  mkdir(new_dir)
```

Leemos cada imagen y le cambiamos el tamaño para mantener la proporción de una hoja corta y que todas midan lo mismo.

[ ]:
```
width = 4000
height = 5090
```

[ ]:
```
n=alumnos[12]
img = cv2.imread(path_planas + 'entrenamiento_' + n + '.png', 0)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
plt.imshow(resized, cmap='gray')
```

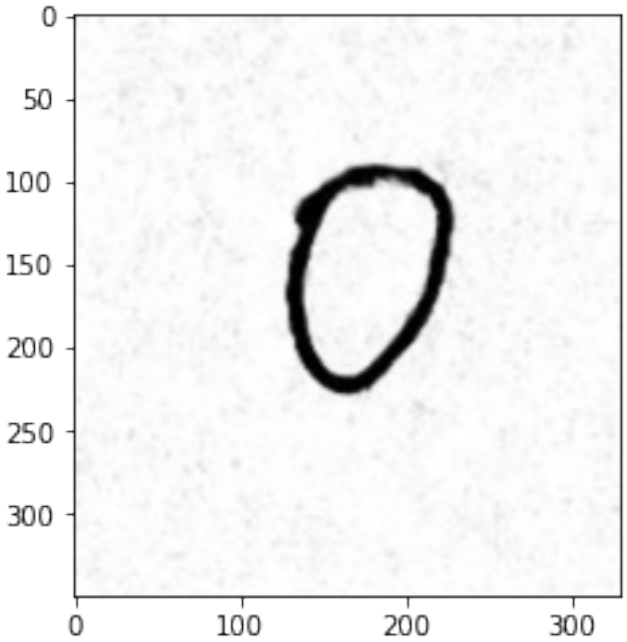
[ ]:
```
<matplotlib.image.AxesImage at 0x7f4ee8878350>
```



Obtenemos el primer número calculando las coordenadas del primer punto.

[ ]:
```
sub_img = resized[1340:1690,230:560]
plt.imshow(sub_img, cmap='gray')
```

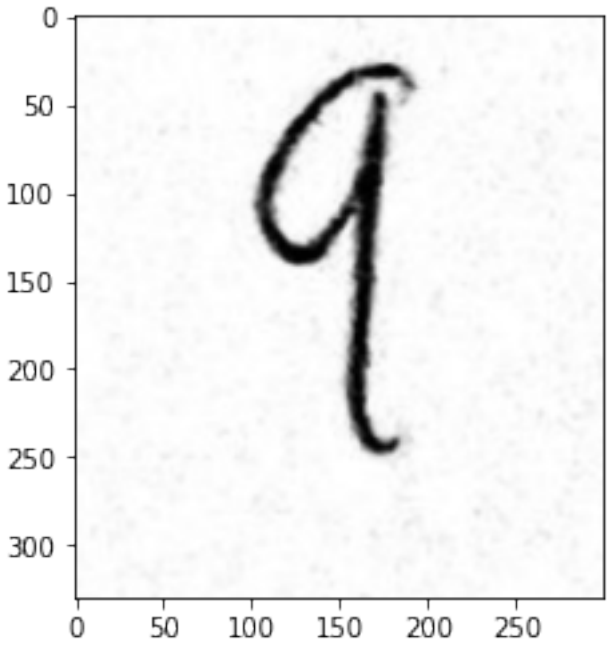
[ ]:
```
<matplotlib.image.AxesImage at 0x7f4ee8269fd0>
```



Iteramos en cada uno de los números para obtener el cambio en x y cambio en y y obtener todos los números de la hoja.

[ ]:
```
i =9
j =9
x = 360
y= 360
sub_img1 = resized[(1350 + y * j):(1680 + y * j),(230 + x * i):(530 + x * i)]
plt.imshow(sub_img1, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f4ee7c79310>
```



Guardamos las coordenadas del primer número y los cambios entre cada uno

[ ]:
```
coordenadas_0 = np.array([[1280,1610],[260,570],[1330, 1660],[260,610],␣
↪[1230,1590], [250,570],[1300,1590],[270,540],
                          [1270,1570],[200,510],␣
↪[1330,1640],[180,480],[1360,1660], [230,560], [1455,1760],[175,500],
                          [1540,1860],[200,530],[1295,1610],[160,500],[␣
↪1480,1810 ],[220,505],[1330,1620],[330,630],
                          [1350,1680], [230,530],
                         ␣
↪[1300,1640],[190,490],[1365,1675],[255,570],[1280,1590],[190,500],[1375,1670],[260,590],
```

```
                          [1355,1650],[195,510],[1290,1620],[130,450],␣
 ↪[1350,1660], [280,600]] )
```

[ ]:
```
cambio_x = [350,350,355,350,365,365,360,370,365,360,360,335,360␣
 ↪,365,355,370,355,360,380,355]
cambio_y = [360,360,370,350,365,365,360,370,360,370,360,340,␣
 ↪360,360,355,370,355,360,370,360]
```

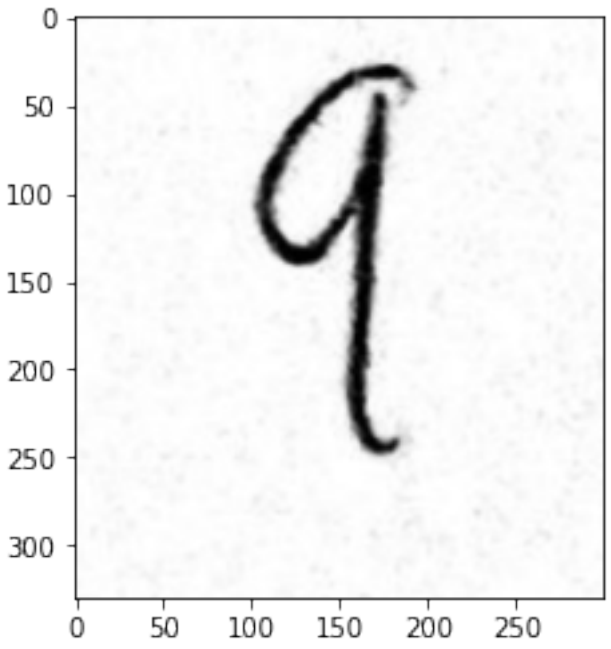
[ ]:
```
coordenadas_0[24]
```

[ ]:
```
array([1350, 1680])
```

Comprobamos que se hayan guardado correctamente.

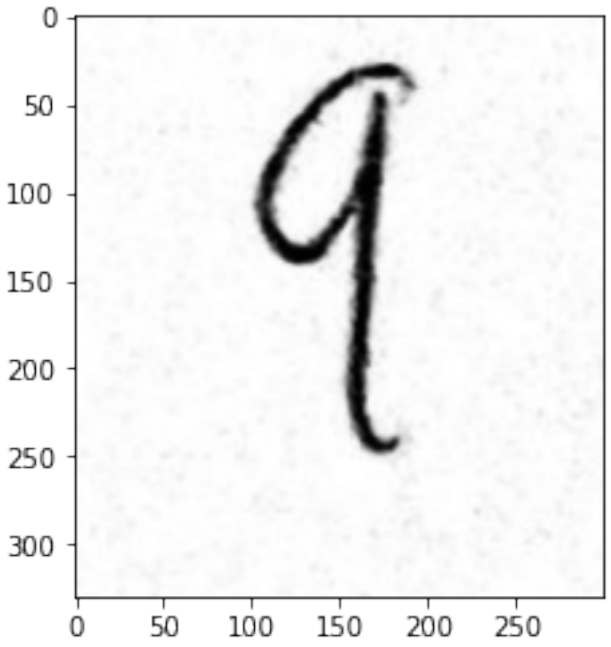
[ ]:
```
i = 9
j = 9
x = cambio_x[12]
y= cambio_y[12]
sub_img1 = resized[(coordenadas_0[24][0] + y * j):(coordenadas_0[24][1] + y *␣
 ↪j),(coordenadas_0[25][0] + x * i):(coordenadas_0[25][1] + x * i)]
plt.imshow(sub_img1, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f4ee7be4190>
```



Extraemos las todas imágenes de la hoja y las guardamos en la carpeta correspondiente de cada número.

[ ]:
```
x = cambio_x[12]
y= cambio_y[12]
coordenadas_x = coordenadas_0[24]
coordenadas_y = coordenadas_0[25]
path1 = root + 'Full_DataSet/'
scan = cv2.imread(path_planas + 'entrenamiento_' + n + '.png', 0)
resized = cv2.resize(scan, dim, interpolation = cv2.INTER_AREA)
for i in range(0,10):
  new_dir = path1 + '/'  + str(i)
  for j in range(0,10):
    #print(i,j)
    sub_img1 = resized[(coordenadas_x[0] + y * j):(coordenadas_x[1] + y *␣
 ↪j),(coordenadas_y[0] + x * i):(coordenadas_y[1] + x * i)]
    #print(new_dir +'/' + alumno + str(j) + '.png')
    cv2.imwrite(new_dir +'/alumno' + n +'_' + str(j) + '.png', sub_img1)
    plt.imshow(sub_img1, cmap='gray')
```



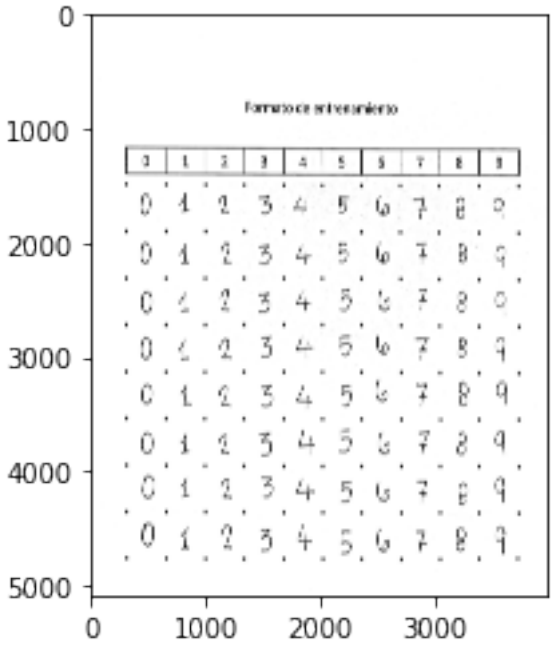
[ ]:
```
path2 = root + 'OOT_DataSet/'
mkdir(path2)
for i in range(0,10):
  new_dir = path2 + '/'  + str(i)
  mkdir(new_dir)
```

[ ]:
```
alumnos = [str(x) for x in range(1,9)]
for n in alumnos:
```

```
  path = path_planas + 'validar_' + n +'.pdf'
  pages = convert_from_path(path, 500)
  for page in pages:
    page.save(path_planas + 'validar_' + n + '.png', 'PNG')
```

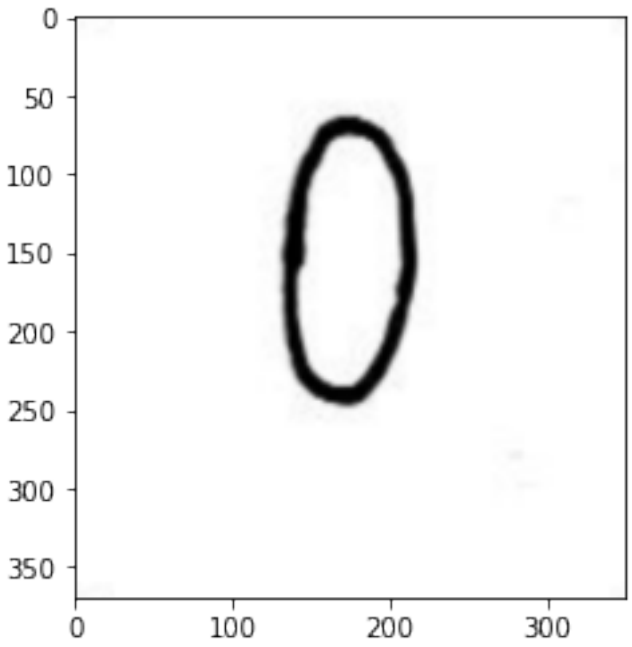
[ ]:
```
width = 4000
height = 5090
n=alumnos[6]
img = cv2.imread(path_planas + 'validar_' + n + '.png', 0)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
plt.imshow(resized, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fb4fd873cd0>
```



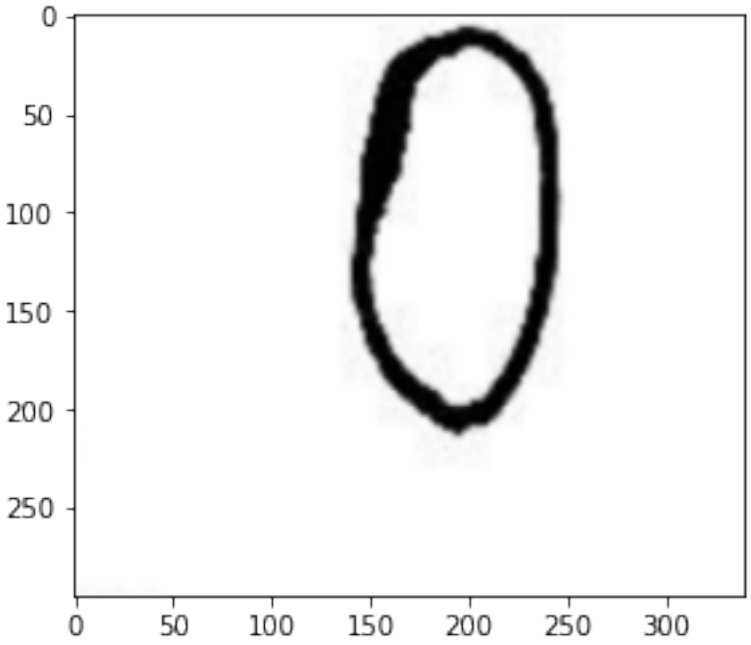
[ ]:
```
sub_img = resized[1510:1880,310:660]
plt.imshow(sub_img, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fb4fd36f550>
```



[ ]:
```
i =0
j =8
x = 345
y= 360
sub_img1 = resized[(1565 + y * j):(1860 + y * j),(310 + x * i):(650 + x * i)]
plt.imshow(sub_img1, cmap='gray')
```

[ ]: <matplotlib.image.AxesImage at 0x7fb4f989de10>

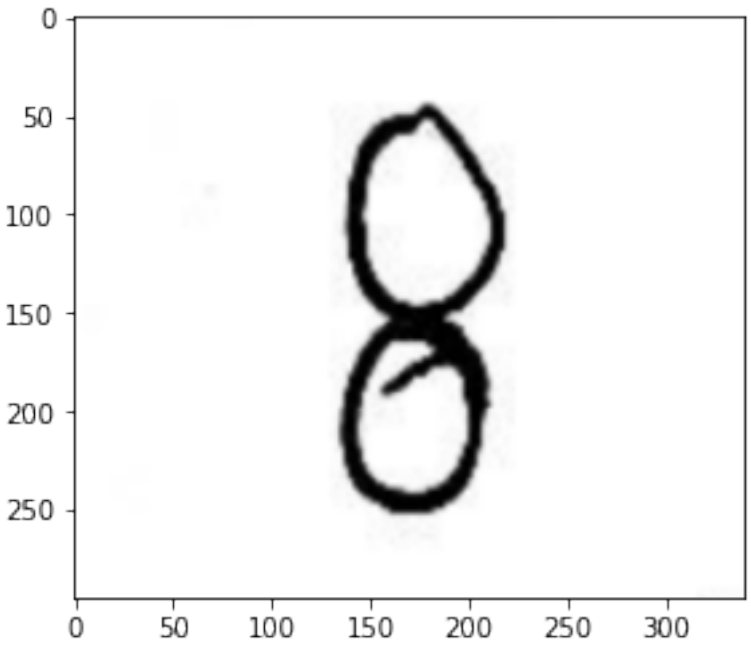


[ ]:
```
coordenadasval_0 = np.array([␣
 ↪[1330,1580],[150,425],[990,1330],[150,490],[1250,1520],[150,480],
                             [1260,1600],[130,440], [1450,1770], [295,550],␣
 ↪[1230,1545], [125,465],
                             [1320,1610],[390,680],[1565,1860],[310,650]])
```

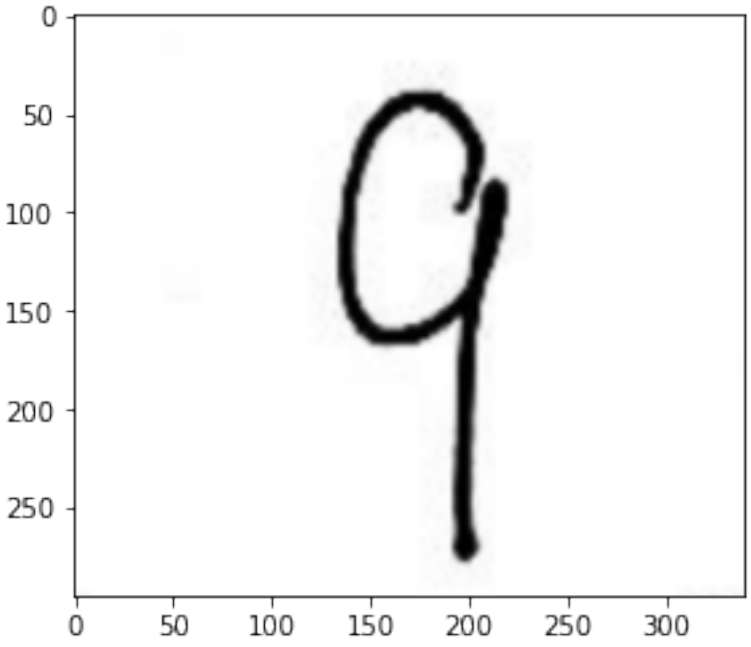
[ ]:
```
cambioval_x = [370,375,370,375,350,380,345]
cambioval_y = [380,390,380,360,350,370,360]
```

[ ]:
```
i = 8
j = 8
x = cambioval_x[6]
y= cambioval_y[6]
sub_img1 = resized[(coordenadasval_0[14][0] + y * j):(coordenadasval_0[14][1] +␣
 ↪y * j),(coordenadasval_0[15][0] + x * i):(coordenadasval_0[15][1] + x * i)]
plt.imshow(sub_img1, cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fb4f980e590>
```



[ ]:
```
x = cambioval_x[6]
y= cambioval_y[6]
coordenadasval_x = coordenadasval_0[14]
coordenadasval_y = coordenadasval_0[15]
path2 = root + 'Val_Set/'
scan = cv2.imread(path_planas + 'validar_' + n + '.png', 0)
resized = cv2.resize(scan, dim, interpolation = cv2.INTER_AREA)
for i in range(0,10):
  new_dir = path2 + '/'  + str(i)
  for j in range(8,9):
    #print(i,j)
    sub_img1 = resized[(coordenadasval_x[0] + y * j):(coordenadasval_x[1] + y *␣
 ↪j),(coordenadasval_y[0] + x * i):(coordenadasval_y[1] + x * i)]
    #print(new_dir +'/' + alumno + str(j) + '.png')
    cv2.imwrite(new_dir +'/alumno' + n +'_' + str(j) + '.png', sub_img1)
    plt.imshow(sub_img1, cmap='gray')
```


0
50
100
150
200
250
0
50
100
150
200
250
300

B. Modelo de la red neuronal

Código para la construcción del modelo de red neuronal convolucional, su entrenamiento y evaluación con los conjuntos de imágenes correspondientes.

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[3]:
```
from tensorflow.keras.models import Sequential
```

[4]:
```
import cv2
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import  InputLayer
from tensorflow.keras import regularizers
import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import time
import pandas as pd
import numpy as np
```

[5]:
```
root = './drive/MyDrive/Entrenamiento/'
```

[6]:
```
BS = 64
alpha = 0.00005
semilla = 1
np.random.seed(semilla)
tensorflow.random.set_seed(semilla)
```

[6]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255.,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split = 0.2
)
```

[7]:
```
path_train = root + 'Train/'
train_set = train_datagen.flow_from_directory(
    path_train,
    target_size=(128,128),
    batch_size=BS,
    class_mode='categorical',
    seed = semilla,
```

```
    subset = 'training',
    color_mode='grayscale'
)
```

```
Found 1250 images belonging to 10 classes.
```

[8]:
```
val_set = train_datagen.flow_from_directory(
    path_train,
    target_size=(128,128),
    batch_size=BS,
    class_mode='categorical',
    seed = semilla,
    subset = 'validation',
    color_mode='grayscale'
)
```

```
Found 310 images belonging to 10 classes.
```

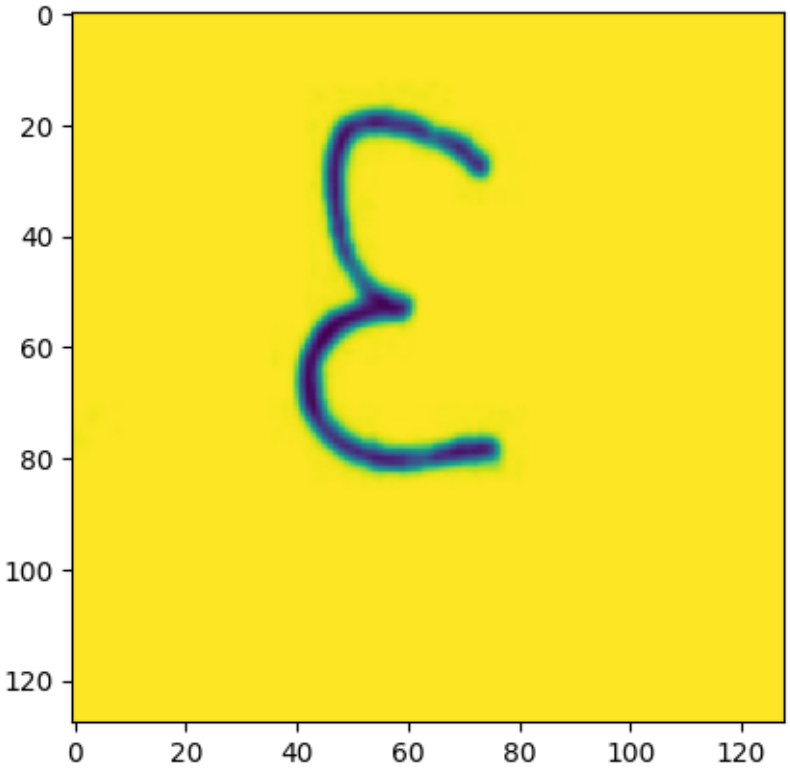
[9]:
```
for x_batch, y_batch in train_set:
    x_train = x_batch
    y_train = y_batch
    break
```

[10]:
```
for x_batch, y_batch in val_set:
    x_val = x_batch
    y_val = y_batch
    break
```

[11]:
```
print('Etiqueta ' ,y_train[0],np.argmax(y_train[0]))
plt.imshow(x_train[0].squeeze(axis=2))
```

```
Etiqueta  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.] 3
```

[11]:
```
<matplotlib.image.AxesImage at 0x7bcb6967dae0>
```



[19]:
```
# Creamos la estructura base con Sequencial de Tensorflow/Keras
cnn = Sequential()

cnn.add(InputLayer(input_shape=(128, 128, 1)))
# Primer capa convolucional
cnn.add(Conv2D(filters=400,kernel_size = 8, strides=2,␣
 ↪input_shape=(128,128,1),activation = 'relu',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Primer capa de Max Pooling
cnn.add(MaxPooling2D(pool_size=(7,7), strides=2))

# Segunda capa convolucional
cnn.add(Conv2D(filters=100,kernel_size = 5, strides= 2, activation = 'relu' ,
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Segunda capa de Max Pooling
```

```
cnn.add(MaxPooling2D(pool_size=(6,6), strides=2))

# Tercera capa convolucional
cnn.add(Conv2D(filters=50,kernel_size = 2,activation = 'relu',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Cuarta capa convolucional
cnn.add(Conv2D(filters=50,kernel_size = 3,activation = 'relu',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Capa de Flattening
cnn.add(Flatten())

# Capa Fully Connected (Primer capa intermedia)
cnn.add(Dense(units = 50,activation = 'relu',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Capa Fully Connected (Segunda capa intermedia)
cnn.add(Dense(units = 25,activation = 'relu',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))

# Capa de salida, para clasificacion
cnn.add(Dense(units = 10,activation = 'softmax',
    kernel_regularizer=regularizers.l1_l2(l1=alpha, l2=alpha),
    bias_regularizer=regularizers.l2(alpha),
    activity_regularizer=regularizers.l2(alpha)))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/input_layer.py:26:
UserWarning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[20]:
```
lr = 0.00005
opt= tensorflow.keras.optimizers.Adam(learning_rate=lr)
```

```
cnn.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics=␣
 ↪['accuracy'])
```

[21]:
```
cnn.summary()
```

```
Model: "sequential"


| Layer (type)                         | Output Shape                |        ␣
 ↪Param # |

| conv2d (Conv2D)                      | (None, 61, 61, 400)         |        ␣
 ↪26,000|
|--------------------------------------|-----------------------------|----------------|
| max_pooling2d (MaxPooling2D)         | (None, 28, 28, 400)         |              ␣
 ↪0|
|--------------------------------------|-----------------------------|----------------|
| conv2d_1 (Conv2D)                    | (None, 12, 12, 100)         |      ␣
 ↪1,000,100|
|--------------------------------------|-----------------------------|----------------|
| max_pooling2d_1 (MaxPooling2D)       | (None, 4, 4, 100)           |              ␣
 ↪0|
|--------------------------------------|-----------------------------|----------------|
| conv2d_2 (Conv2D)                    | (None, 3, 3, 50)            |        ␣
 ↪20,050|
|--------------------------------------|-----------------------------|----------------|
| conv2d_3 (Conv2D)                    | (None, 1, 1, 50)            |        ␣
 ↪22,550|
|--------------------------------------|-----------------------------|----------------|
| flatten (Flatten)                    | (None, 50)                  |              ␣
 ↪0|
|--------------------------------------|-----------------------------|----------------|
| dense (Dense)                        | (None, 50)                  |          ␣
 ↪2,550|
|--------------------------------------|-----------------------------|----------------|
| dense_1 (Dense)                      | (None, 25)                  |          ␣
 ↪1,275|
|--------------------------------------|-----------------------------|----------------|
| dense_2 (Dense)                      | (None, 10)                  |           ␣
 ↪260|
|--------------------------------------|-----------------------------|----------------|


 Total params: 1,072,785 (4.09 MB)


 Trainable params: 1,072,785 (4.09 MB)
```

```
 Non-trainable params: 0 (0.00 B)
```

[22]:
```
T_Epochs =  150
tensorflow.random.set_seed(semilla)
st= time.time()
history = cnn.fit(
    train_set,
    steps_per_epoch = len(train_set)-1,
    epochs = T_Epochs,
    validation_data = val_set,
    validation_steps= len(val_set)-1,
    verbose = 1
)
et = time.time()
print('Tiempo de entrenamiento: ',et-st)
```

```
Epoch 1/150

/usr/local/lib/python3.10/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

19/19 ---------------- 690s 26s/step -
accuracy: 0.1107 - loss: 11.4577 - val_accuracy: 0.1211 - val_loss: 5.7860
Epoch 2/150
 1/19 ----------------- 1:38 5s/step -
accuracy: 0.1406 - loss: 5.7801

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()`
function when building your dataset.
  self.gen.throw(typ, value, traceback)

19/19 ---------------- 25s 1s/step -
accuracy: 0.1406 - loss: 5.7801 - val_accuracy: 0.0741 - val_loss: 5.1650
Epoch 3/150
19/19 ---------------- 168s 7s/step -
accuracy: 0.1091 - loss: 4.8430 - val_accuracy: 0.1094 - val_loss: 3.4844
Epoch 4/150
19/19 ---------------- 6s 59ms/step -
accuracy: 0.1406 - loss: 3.4844 - val_accuracy: 0.0926 - val_loss: 3.3607
Epoch 5/150
19/19 ---------------- 127s 6s/step -
```

```
accuracy: 0.1184 - loss: 3.2897 - val_accuracy: 0.1328 - val_loss: 3.0104
Epoch 6/150
19/19 ---------- 14s 381ms/step -
accuracy: 0.2031 - loss: 3.0103 - val_accuracy: 0.1481 - val_loss: 2.9819
Epoch 7/150
19/19 ---------- 126s 6s/step -
accuracy: 0.1684 - loss: 2.9667 - val_accuracy: 0.1133 - val_loss: 2.8947
Epoch 8/150
19/19 ---------- 9s 59ms/step -
accuracy: 0.1562 - loss: 2.8959 - val_accuracy: 0.0926 - val_loss: 2.8873
Epoch 9/150
19/19 ---------- 133s 7s/step -
accuracy: 0.2213 - loss: 2.8809 - val_accuracy: 0.2266 - val_loss: 2.8485
Epoch 10/150
19/19 ---------- 7s 59ms/step -
accuracy: 0.2656 - loss: 2.8475 - val_accuracy: 0.1852 - val_loss: 2.8451
Epoch 11/150
19/19 ---------- 143s 7s/step -
accuracy: 0.2754 - loss: 2.8395 - val_accuracy: 0.2500 - val_loss: 2.8146
Epoch 12/150
19/19 ---------- 10s 59ms/step -
accuracy: 0.3125 - loss: 2.8125 - val_accuracy: 0.2222 - val_loss: 2.8085
Epoch 13/150
19/19 ---------- 127s 7s/step -
accuracy: 0.2811 - loss: 2.8054 - val_accuracy: 0.2148 - val_loss: 2.7824
Epoch 14/150
19/19 ---------- 13s 429ms/step -
accuracy: 0.2500 - loss: 2.7805 - val_accuracy: 0.2593 - val_loss: 2.7766
Epoch 15/150
19/19 ---------- 132s 7s/step -
accuracy: 0.2966 - loss: 2.7726 - val_accuracy: 0.2500 - val_loss: 2.7509
Epoch 16/150
19/19 ---------- 6s 72ms/step -
accuracy: 0.2647 - loss: 2.7359 - val_accuracy: 0.2407 - val_loss: 2.7434
Epoch 17/150
19/19 ---------- 136s 7s/step -
accuracy: 0.2994 - loss: 2.7382 - val_accuracy: 0.2539 - val_loss: 2.7203
Epoch 18/150
19/19 ---------- 8s 62ms/step -
accuracy: 0.2500 - loss: 2.7166 - val_accuracy: 0.3333 - val_loss: 2.7137
Epoch 19/150
19/19 ---------- 127s 7s/step -
accuracy: 0.3114 - loss: 2.7040 - val_accuracy: 0.2930 - val_loss: 2.6875
Epoch 20/150
19/19 ---------- 13s 456ms/step -
accuracy: 0.3125 - loss: 2.6841 - val_accuracy: 0.2593 - val_loss: 2.6949
Epoch 21/150
19/19 ---------- 138s 7s/step -
```

```
accuracy: 0.3243 - loss: 2.6727 - val_accuracy: 0.2852 - val_loss: 2.6612
Epoch 22/150
19/19 ---------- 7s 61ms/step -
accuracy: 0.3281 - loss: 2.6429 - val_accuracy: 0.2963 - val_loss: 2.6482
Epoch 23/150
19/19 ---------- 183s 7s/step -
accuracy: 0.3051 - loss: 2.6420 - val_accuracy: 0.2930 - val_loss: 2.6295
Epoch 24/150
19/19 ---------- 6s 59ms/step -
accuracy: 0.4375 - loss: 2.5841 - val_accuracy: 0.2037 - val_loss: 2.6291
Epoch 25/150
19/19 ---------- 128s 7s/step -
accuracy: 0.3014 - loss: 2.6027 - val_accuracy: 0.3047 - val_loss: 2.5964
Epoch 26/150
19/19 ---------- 9s 59ms/step -
accuracy: 0.3438 - loss: 2.5894 - val_accuracy: 0.3333 - val_loss: 2.6189
Epoch 27/150
19/19 ---------- 132s 6s/step -
accuracy: 0.3308 - loss: 2.5615 - val_accuracy: 0.2891 - val_loss: 2.5669
Epoch 28/150
19/19 ---------- 9s 60ms/step -
accuracy: 0.2969 - loss: 2.5587 - val_accuracy: 0.3333 - val_loss: 2.5555
Epoch 29/150
19/19 ---------- 133s 7s/step -
accuracy: 0.3425 - loss: 2.5257 - val_accuracy: 0.2773 - val_loss: 2.5427
Epoch 30/150
19/19 ---------- 8s 59ms/step -
accuracy: 0.3281 - loss: 2.4731 - val_accuracy: 0.3333 - val_loss: 2.4821
Epoch 31/150
19/19 ---------- 126s 7s/step -
accuracy: 0.3181 - loss: 2.4864 - val_accuracy: 0.3125 - val_loss: 2.5027
Epoch 32/150
19/19 ---------- 6s 59ms/step -
accuracy: 0.3438 - loss: 2.4588 - val_accuracy: 0.3148 - val_loss: 2.4884
Epoch 33/150
19/19 ---------- 134s 6s/step -
accuracy: 0.3538 - loss: 2.4447 - val_accuracy: 0.3320 - val_loss: 2.4645
Epoch 34/150
19/19 ---------- 14s 459ms/step -
accuracy: 0.2656 - loss: 2.4137 - val_accuracy: 0.3148 - val_loss: 2.4388
Epoch 35/150
19/19 ---------- 127s 7s/step -
accuracy: 0.3598 - loss: 2.3837 - val_accuracy: 0.3555 - val_loss: 2.4340
Epoch 36/150
19/19 ---------- 8s 59ms/step -
accuracy: 0.3750 - loss: 2.3897 - val_accuracy: 0.3333 - val_loss: 2.4336
Epoch 37/150
19/19 ---------- 144s 7s/step -
```

```
accuracy: 0.3932 - loss: 2.3458 - val_accuracy: 0.3320 - val_loss: 2.4145
Epoch 38/150
19/19 -------------- 14s 464ms/step -
accuracy: 0.4062 - loss: 2.3558 - val_accuracy: 0.4630 - val_loss: 2.3181
Epoch 39/150
19/19 -------------- 183s 7s/step -
accuracy: 0.4126 - loss: 2.3023 - val_accuracy: 0.3242 - val_loss: 2.3715
Epoch 40/150
19/19 -------------- 5s 59ms/step -
accuracy: 0.4118 - loss: 2.1928 - val_accuracy: 0.3333 - val_loss: 2.3196
Epoch 41/150
19/19 -------------- 134s 6s/step -
accuracy: 0.3857 - loss: 2.2687 - val_accuracy: 0.3555 - val_loss: 2.3591
Epoch 42/150
19/19 -------------- 7s 60ms/step -
accuracy: 0.4375 - loss: 2.2221 - val_accuracy: 0.4444 - val_loss: 2.2620
Epoch 43/150
19/19 -------------- 134s 7s/step -
accuracy: 0.4189 - loss: 2.2195 - val_accuracy: 0.3359 - val_loss: 2.3298
Epoch 44/150
19/19 -------------- 6s 59ms/step -
accuracy: 0.4531 - loss: 2.0923 - val_accuracy: 0.4259 - val_loss: 2.1937
Epoch 45/150
19/19 -------------- 135s 7s/step -
accuracy: 0.4440 - loss: 2.1684 - val_accuracy: 0.3594 - val_loss: 2.2998
Epoch 46/150
19/19 -------------- 6s 59ms/step -
accuracy: 0.4844 - loss: 2.1130 - val_accuracy: 0.5000 - val_loss: 2.1370
Epoch 47/150
19/19 -------------- 126s 6s/step -
accuracy: 0.4470 - loss: 2.1487 - val_accuracy: 0.3633 - val_loss: 2.2529
Epoch 48/150
19/19 -------------- 9s 60ms/step -
accuracy: 0.5000 - loss: 2.1045 - val_accuracy: 0.4815 - val_loss: 2.1614
Epoch 49/150
19/19 -------------- 133s 7s/step -
accuracy: 0.4466 - loss: 2.1045 - val_accuracy: 0.3789 - val_loss: 2.2283
Epoch 50/150
19/19 -------------- 9s 60ms/step -
accuracy: 0.4375 - loss: 2.0330 - val_accuracy: 0.3889 - val_loss: 2.1737
Epoch 51/150
19/19 -------------- 134s 6s/step -
accuracy: 0.4882 - loss: 2.0809 - val_accuracy: 0.4141 - val_loss: 2.1884
Epoch 52/150
19/19 -------------- 7s 59ms/step -
accuracy: 0.3438 - loss: 2.1784 - val_accuracy: 0.4259 - val_loss: 2.2237
Epoch 53/150
19/19 -------------- 128s 7s/step -
```

```
accuracy: 0.4690 - loss: 2.0528 - val_accuracy: 0.3945 - val_loss: 2.1752
Epoch 54/150
19/19 ---------- 16s 64ms/step -
accuracy: 0.5625 - loss: 1.9851 - val_accuracy: 0.4074 - val_loss: 2.1465
Epoch 55/150
19/19 ---------- 184s 7s/step -
accuracy: 0.5015 - loss: 2.0185 - val_accuracy: 0.4141 - val_loss: 2.1639
Epoch 56/150
19/19 ---------- 6s 59ms/step -
accuracy: 0.6562 - loss: 1.9013 - val_accuracy: 0.3704 - val_loss: 2.0926
Epoch 57/150
19/19 ---------- 126s 6s/step -
accuracy: 0.5495 - loss: 1.9628 - val_accuracy: 0.4375 - val_loss: 2.1168
Epoch 58/150
19/19 ---------- 8s 59ms/step -
accuracy: 0.5781 - loss: 2.0043 - val_accuracy: 0.4074 - val_loss: 2.1649
Epoch 59/150
19/19 ---------- 133s 6s/step -
accuracy: 0.5181 - loss: 1.9712 - val_accuracy: 0.4688 - val_loss: 2.0898
Epoch 60/150
19/19 ---------- 8s 59ms/step -
accuracy: 0.5625 - loss: 1.9264 - val_accuracy: 0.4074 - val_loss: 2.1207
Epoch 61/150
19/19 ---------- 142s 7s/step -
accuracy: 0.5677 - loss: 1.9212 - val_accuracy: 0.4688 - val_loss: 2.0907
Epoch 62/150
19/19 ---------- 8s 60ms/step -
accuracy: 0.6719 - loss: 1.8509 - val_accuracy: 0.4444 - val_loss: 2.0397
Epoch 63/150
19/19 ---------- 187s 6s/step -
accuracy: 0.5984 - loss: 1.8818 - val_accuracy: 0.5000 - val_loss: 2.0472
Epoch 64/150
19/19 ---------- 14s 430ms/step -
accuracy: 0.5156 - loss: 2.0002 - val_accuracy: 0.4074 - val_loss: 2.0965
Epoch 65/150
19/19 ---------- 126s 6s/step -
accuracy: 0.6096 - loss: 1.8474 - val_accuracy: 0.4766 - val_loss: 2.0257
Epoch 66/150
19/19 ---------- 9s 88ms/step -
accuracy: 0.7188 - loss: 1.7777 - val_accuracy: 0.5185 - val_loss: 1.9625
Epoch 67/150
19/19 ---------- 126s 7s/step -
accuracy: 0.6154 - loss: 1.8207 - val_accuracy: 0.4922 - val_loss: 1.9984
Epoch 68/150
19/19 ---------- 6s 60ms/step -
accuracy: 0.5938 - loss: 1.8737 - val_accuracy: 0.5370 - val_loss: 1.9293
Epoch 69/150
19/19 ---------- 132s 7s/step -
```

```
accuracy: 0.6464 - loss: 1.7648 - val_accuracy: 0.4922 - val_loss: 2.0199
Epoch 70/150
19/19 ---------------- 12s 60ms/step -
accuracy: 0.5781 - loss: 1.8761 - val_accuracy: 0.5556 - val_loss: 1.8992
Epoch 71/150
19/19 ---------------- 128s 7s/step -
accuracy: 0.6452 - loss: 1.7470 - val_accuracy: 0.5078 - val_loss: 1.9834
Epoch 72/150
19/19 ---------------- 9s 60ms/step -
accuracy: 0.5781 - loss: 1.8255 - val_accuracy: 0.4630 - val_loss: 1.8582
Epoch 73/150
19/19 ---------------- 133s 7s/step -
accuracy: 0.6642 - loss: 1.7182 - val_accuracy: 0.5547 - val_loss: 1.8948
Epoch 74/150
19/19 ---------------- 9s 87ms/step -
accuracy: 0.5938 - loss: 1.8562 - val_accuracy: 0.4630 - val_loss: 2.0446
Epoch 75/150
19/19 ---------------- 132s 6s/step -
accuracy: 0.6471 - loss: 1.7220 - val_accuracy: 0.5664 - val_loss: 1.9006
Epoch 76/150
19/19 ---------------- 9s 60ms/step -
accuracy: 0.7031 - loss: 1.6785 - val_accuracy: 0.5185 - val_loss: 1.9323
Epoch 77/150
19/19 ---------------- 129s 7s/step -
accuracy: 0.6660 - loss: 1.6890 - val_accuracy: 0.5391 - val_loss: 1.8974
Epoch 78/150
19/19 ---------------- 4s 60ms/step -
accuracy: 0.7941 - loss: 1.4187 - val_accuracy: 0.4630 - val_loss: 1.9230
Epoch 79/150
19/19 ---------------- 146s 7s/step -
accuracy: 0.6813 - loss: 1.6396 - val_accuracy: 0.5352 - val_loss: 1.9106
Epoch 80/150
19/19 ---------------- 8s 59ms/step -
accuracy: 0.7656 - loss: 1.5966 - val_accuracy: 0.6481 - val_loss: 1.7213
Epoch 81/150
19/19 ---------------- 126s 7s/step -
accuracy: 0.6924 - loss: 1.6422 - val_accuracy: 0.5273 - val_loss: 1.8964
Epoch 82/150
19/19 ---------------- 14s 466ms/step -
accuracy: 0.6406 - loss: 1.6789 - val_accuracy: 0.6111 - val_loss: 1.7399
Epoch 83/150
19/19 ---------------- 128s 7s/step -
accuracy: 0.7096 - loss: 1.6063 - val_accuracy: 0.5820 - val_loss: 1.8508
Epoch 84/150
19/19 ---------------- 8s 59ms/step -
accuracy: 0.7969 - loss: 1.5368 - val_accuracy: 0.5741 - val_loss: 1.7633
Epoch 85/150
19/19 ---------------- 126s 7s/step -
```

```
accuracy: 0.7155 - loss: 1.5798 - val_accuracy: 0.5938 - val_loss: 1.8564
Epoch 86/150
19/19 ---------------- 13s 409ms/step -
accuracy: 0.7188 - loss: 1.6195 - val_accuracy: 0.5926 - val_loss: 1.7218
Epoch 87/150
19/19 ---------------- 139s 7s/step -
accuracy: 0.7138 - loss: 1.5764 - val_accuracy: 0.6055 - val_loss: 1.7868
Epoch 88/150
19/19 ---------------- 7s 60ms/step -
accuracy: 0.8125 - loss: 1.5688 - val_accuracy: 0.5926 - val_loss: 1.8282
Epoch 89/150
19/19 ---------------- 124s 6s/step -
accuracy: 0.7252 - loss: 1.5642 - val_accuracy: 0.5898 - val_loss: 1.8087
Epoch 90/150
19/19 ---------------- 15s 471ms/step -
accuracy: 0.7812 - loss: 1.5254 - val_accuracy: 0.6481 - val_loss: 1.7402
Epoch 91/150
19/19 ---------------- 127s 7s/step -
accuracy: 0.7596 - loss: 1.5226 - val_accuracy: 0.6250 - val_loss: 1.7926
Epoch 92/150
19/19 ---------------- 9s 60ms/step -
accuracy: 0.8281 - loss: 1.4406 - val_accuracy: 0.6667 - val_loss: 1.7265
Epoch 93/150
19/19 ---------------- 127s 7s/step -
accuracy: 0.7693 - loss: 1.5019 - val_accuracy: 0.6172 - val_loss: 1.7792
Epoch 94/150
19/19 ---------------- 6s 60ms/step -
accuracy: 0.8125 - loss: 1.3935 - val_accuracy: 0.5926 - val_loss: 1.7367
Epoch 95/150
19/19 ---------------- 136s 6s/step -
accuracy: 0.7820 - loss: 1.4686 - val_accuracy: 0.5898 - val_loss: 1.7929
Epoch 96/150
19/19 ---------------- 9s 64ms/step -
accuracy: 0.7812 - loss: 1.5288 - val_accuracy: 0.6667 - val_loss: 1.7184
Epoch 97/150
19/19 ---------------- 135s 7s/step -
accuracy: 0.7890 - loss: 1.4682 - val_accuracy: 0.6250 - val_loss: 1.7831
Epoch 98/150
19/19 ---------------- 8s 59ms/step -
accuracy: 0.8281 - loss: 1.3998 - val_accuracy: 0.6481 - val_loss: 1.6125
Epoch 99/150
19/19 ---------------- 131s 6s/step -
accuracy: 0.8067 - loss: 1.4351 - val_accuracy: 0.6523 - val_loss: 1.7581
Epoch 100/150
19/19 ---------------- 8s 60ms/step -
accuracy: 0.7656 - loss: 1.5065 - val_accuracy: 0.6667 - val_loss: 1.7340
Epoch 101/150
19/19 ---------------- 134s 7s/step -
```

```
accuracy: 0.8181 - loss: 1.4301 - val_accuracy: 0.6250 - val_loss: 1.7346
Epoch 102/150
19/19 -------------- 7s 60ms/step -
accuracy: 0.8438 - loss: 1.3558 - val_accuracy: 0.6481 - val_loss: 1.6861
Epoch 103/150
19/19 -------------- 126s 6s/step -
accuracy: 0.7971 - loss: 1.4191 - val_accuracy: 0.6562 - val_loss: 1.7197
Epoch 104/150
19/19 -------------- 14s 423ms/step -
accuracy: 0.8281 - loss: 1.3497 - val_accuracy: 0.6481 - val_loss: 1.6675
Epoch 105/150
19/19 -------------- 127s 7s/step -
accuracy: 0.8406 - loss: 1.3742 - val_accuracy: 0.6523 - val_loss: 1.7286
Epoch 106/150
19/19 -------------- 9s 76ms/step -
accuracy: 0.8906 - loss: 1.3310 - val_accuracy: 0.7407 - val_loss: 1.5477
Epoch 107/150
19/19 -------------- 134s 7s/step -
accuracy: 0.8551 - loss: 1.3370 - val_accuracy: 0.6836 - val_loss: 1.6650
Epoch 108/150
19/19 -------------- 9s 60ms/step -
accuracy: 0.8438 - loss: 1.3702 - val_accuracy: 0.6111 - val_loss: 1.7565
Epoch 109/150
19/19 -------------- 133s 7s/step -
accuracy: 0.8610 - loss: 1.3267 - val_accuracy: 0.6641 - val_loss: 1.7262
Epoch 110/150
19/19 -------------- 14s 390ms/step -
accuracy: 0.8594 - loss: 1.3084 - val_accuracy: 0.7037 - val_loss: 1.5924
Epoch 111/150
19/19 -------------- 132s 7s/step -
accuracy: 0.8677 - loss: 1.3366 - val_accuracy: 0.6836 - val_loss: 1.6376
Epoch 112/150
19/19 -------------- 6s 60ms/step -
accuracy: 0.8750 - loss: 1.3037 - val_accuracy: 0.6296 - val_loss: 1.6885
Epoch 113/150
19/19 -------------- 128s 7s/step -
accuracy: 0.8603 - loss: 1.3156 - val_accuracy: 0.6758 - val_loss: 1.6581
Epoch 114/150
19/19 -------------- 9s 61ms/step -
accuracy: 0.8281 - loss: 1.3573 - val_accuracy: 0.6852 - val_loss: 1.5910
Epoch 115/150
19/19 -------------- 136s 7s/step -
accuracy: 0.8455 - loss: 1.3001 - val_accuracy: 0.6758 - val_loss: 1.6605
Epoch 116/150
19/19 -------------- 7s 60ms/step -
accuracy: 0.8906 - loss: 1.3217 - val_accuracy: 0.7037 - val_loss: 1.6444
Epoch 117/150
19/19 -------------- 132s 7s/step -
```

```
accuracy: 0.8604 - loss: 1.2893 - val_accuracy: 0.6953 - val_loss: 1.6301
Epoch 118/150
19/19 -------------- 8s 60ms/step -
accuracy: 0.8438 - loss: 1.3495 - val_accuracy: 0.6296 - val_loss: 1.6558
Epoch 119/150
19/19 -------------- 134s 7s/step -
accuracy: 0.8624 - loss: 1.2800 - val_accuracy: 0.6992 - val_loss: 1.6305
Epoch 120/150
19/19 -------------- 14s 384ms/step -
accuracy: 0.8594 - loss: 1.2766 - val_accuracy: 0.6481 - val_loss: 1.5668
Epoch 121/150
19/19 -------------- 189s 7s/step -
accuracy: 0.9020 - loss: 1.2512 - val_accuracy: 0.6914 - val_loss: 1.6216
Epoch 122/150
19/19 -------------- 9s 60ms/step -
accuracy: 0.8906 - loss: 1.2731 - val_accuracy: 0.7593 - val_loss: 1.5163
Epoch 123/150
19/19 -------------- 127s 7s/step -
accuracy: 0.8824 - loss: 1.2572 - val_accuracy: 0.6992 - val_loss: 1.6104
Epoch 124/150
19/19 -------------- 7s 99ms/step -
accuracy: 0.8750 - loss: 1.2704 - val_accuracy: 0.6667 - val_loss: 1.6522
Epoch 125/150
19/19 -------------- 136s 7s/step -
accuracy: 0.8811 - loss: 1.2382 - val_accuracy: 0.6992 - val_loss: 1.5994
Epoch 126/150
19/19 -------------- 14s 438ms/step -
accuracy: 0.8594 - loss: 1.2536 - val_accuracy: 0.6667 - val_loss: 1.5163
Epoch 127/150
19/19 -------------- 186s 7s/step -
accuracy: 0.8834 - loss: 1.2282 - val_accuracy: 0.6758 - val_loss: 1.6005
Epoch 128/150
19/19 -------------- 6s 59ms/step -
accuracy: 0.7969 - loss: 1.2937 - val_accuracy: 0.6296 - val_loss: 1.5723
Epoch 129/150
19/19 -------------- 135s 6s/step -
accuracy: 0.8951 - loss: 1.2114 - val_accuracy: 0.6914 - val_loss: 1.5841
Epoch 130/150
19/19 -------------- 14s 463ms/step -
accuracy: 0.8750 - loss: 1.2146 - val_accuracy: 0.7222 - val_loss: 1.5157
Epoch 131/150
19/19 -------------- 126s 7s/step -
accuracy: 0.9052 - loss: 1.1677 - val_accuracy: 0.7070 - val_loss: 1.5773
Epoch 132/150
19/19 -------------- 14s 443ms/step -
accuracy: 0.9062 - loss: 1.1923 - val_accuracy: 0.7222 - val_loss: 1.4606
Epoch 133/150
19/19 -------------- 126s 6s/step -
```

```
accuracy: 0.9047 - loss: 1.1758 - val_accuracy: 0.6562 - val_loss: 1.6823
Epoch 134/150
19/19 ---------------- 8s 60ms/step -
accuracy: 0.9219 - loss: 1.1233 - val_accuracy: 0.7037 - val_loss: 1.5695
Epoch 135/150
19/19 ---------------- 128s 7s/step -
accuracy: 0.9073 - loss: 1.1774 - val_accuracy: 0.6602 - val_loss: 1.6272
Epoch 136/150
19/19 ---------------- 6s 58ms/step -
accuracy: 0.8906 - loss: 1.2125 - val_accuracy: 0.7222 - val_loss: 1.4516
Epoch 137/150
19/19 ---------------- 124s 6s/step -
accuracy: 0.9104 - loss: 1.1582 - val_accuracy: 0.6719 - val_loss: 1.6337
Epoch 138/150
19/19 ---------------- 8s 92ms/step -
accuracy: 0.8906 - loss: 1.2331 - val_accuracy: 0.6852 - val_loss: 1.5080
Epoch 139/150
19/19 ---------------- 136s 7s/step -
accuracy: 0.9041 - loss: 1.1843 - val_accuracy: 0.6719 - val_loss: 1.6274
Epoch 140/150
19/19 ---------------- 8s 59ms/step -
accuracy: 0.8438 - loss: 1.2023 - val_accuracy: 0.7778 - val_loss: 1.3197
Epoch 141/150
19/19 ---------------- 136s 7s/step -
accuracy: 0.9227 - loss: 1.1362 - val_accuracy: 0.6953 - val_loss: 1.6042
Epoch 142/150
19/19 ---------------- 7s 59ms/step -
accuracy: 0.8750 - loss: 1.1809 - val_accuracy: 0.6667 - val_loss: 1.5376
Epoch 143/150
19/19 ---------------- 132s 7s/step -
accuracy: 0.9082 - loss: 1.1356 - val_accuracy: 0.6953 - val_loss: 1.5634
Epoch 144/150
19/19 ---------------- 6s 59ms/step -
accuracy: 0.9531 - loss: 1.1270 - val_accuracy: 0.6667 - val_loss: 1.4720
Epoch 145/150
19/19 ---------------- 137s 7s/step -
accuracy: 0.9117 - loss: 1.1321 - val_accuracy: 0.7031 - val_loss: 1.5382
Epoch 146/150
19/19 ---------------- 14s 469ms/step -
accuracy: 0.9219 - loss: 1.0687 - val_accuracy: 0.6667 - val_loss: 1.6994
Epoch 147/150
19/19 ---------------- 126s 6s/step -
accuracy: 0.9114 - loss: 1.1028 - val_accuracy: 0.7109 - val_loss: 1.5466
Epoch 148/150
19/19 ---------------- 8s 69ms/step -
accuracy: 0.9688 - loss: 1.0462 - val_accuracy: 0.6667 - val_loss: 1.4491
Epoch 149/150
19/19 ---------------- 134s 7s/step -
```

```
accuracy: 0.9255 - loss: 1.1026 - val_accuracy: 0.7070 - val_loss: 1.5698
Epoch 150/150
19/19 -------------- 8s 60ms/step -
accuracy: 0.9375 - loss: 1.0890 - val_accuracy: 0.8148 - val_loss: 1.3683
Tiempo de entrenamiento:  11516.948464632034
```

[23]:
```
cnn.save('./drive/MyDrive/Modelo/'+'CNN01.keras')
```

[25]:
```
import json
```

[30]:
```
with open('./drive/MyDrive/Modelo/history_CNN01.json', 'w') as f:
    json.dump(history.history, f)
```

[8]:
```
test_datagen = ImageDataGenerator(
    rescale=1./255.
)
path_test = root + 'Test/'
test_set = test_datagen.flow_from_directory(
    path_test,
    target_size=(128,128),
    class_mode='categorical',
    batch_size=BS,
    seed = semilla,
    color_mode='grayscale'
)
```

```
Found 390 images belonging to 10 classes.
```

[9]:
```
for x_batch, y_batch in test_set:
    x_test = x_batch
    y_test = y_batch
    break
```

[ ]:
```
x_test, y_test = next(test_set)
```

[12]:
```
results_test = cnn.predict(x_test)
results_test_num = np.argmax(results_test,axis=1)
y_test_num = np.argmax(y_test, axis=1)
```

```
2/2 -------------- 1s 480ms/step
```

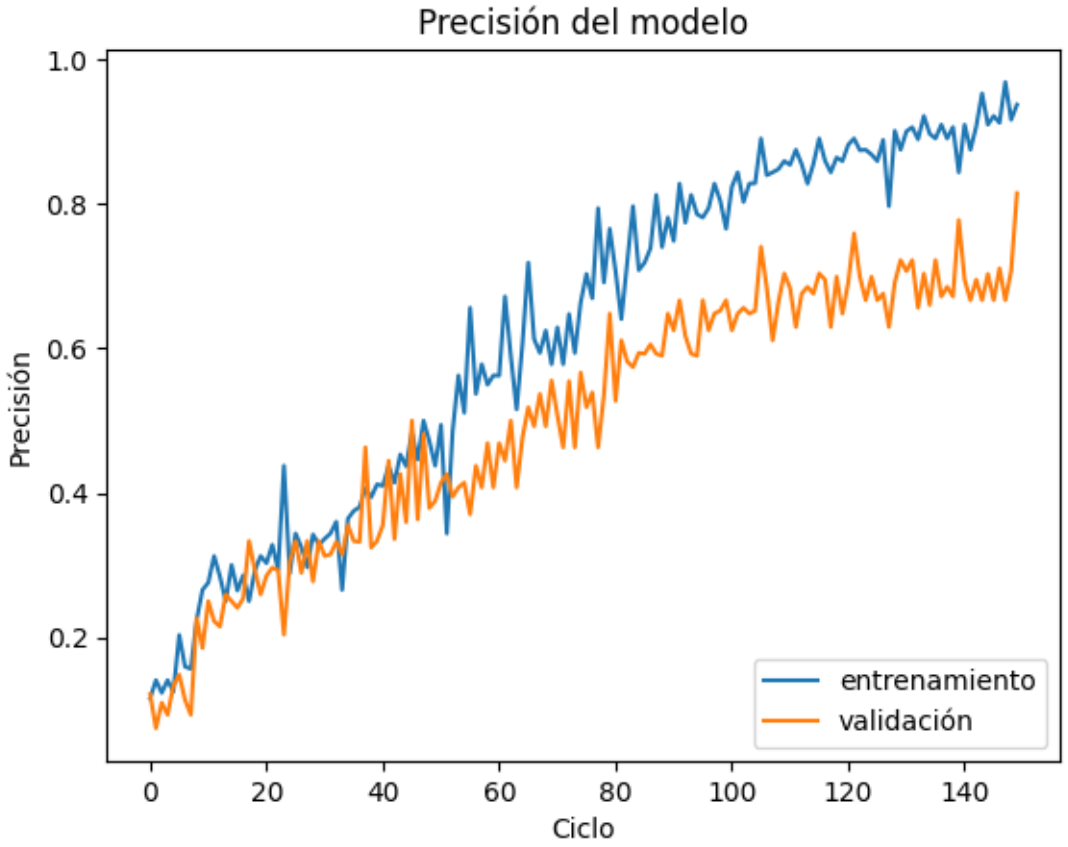
[13]:
```
pd.crosstab(y_test_num, results_test_num, rownames=['Real'],␣
 ↪colnames=['Predicho'])
```

[13]:

Predicho	0	1	2	3	4	5	6	7	8	9
Real										
0	5	0	0	0	0	0	0	1	0	0
1	0	2	0	0	1	0	0	0	0	0
2	0	0	4	0	0	0	0	0	0	0

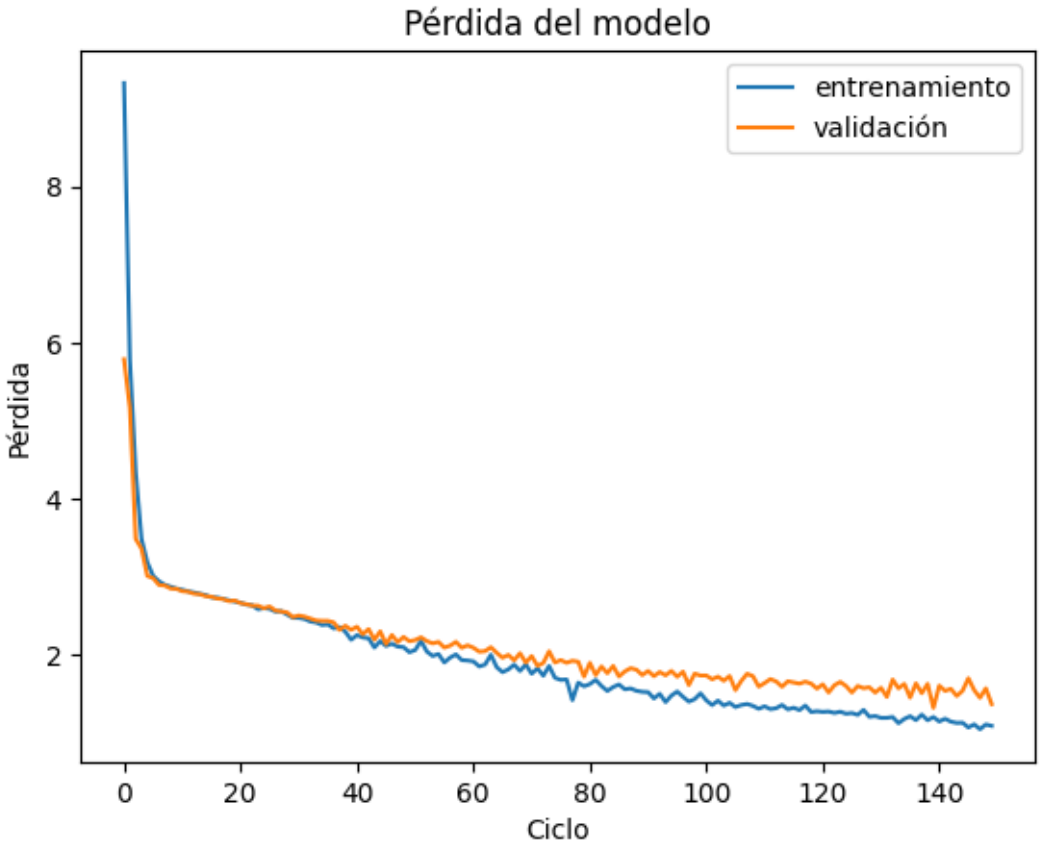
```
3          0   0   1   4    0   0   0   0   0   0
4          0   0   0   0   10   0   0   1   0   0
5          0   0   0   0    0   9   0   0   0   0
6          3   0   0   0    0   0   5   0   0   0
7          0   1   0   0    0   0   0   5   0   0
8          0   0   0   0    0   0   0   1   6   0
9          0   0   0   0    0   0   0   1   0   4
```

[26]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Precisión del modelo')
plt.ylabel('Precisión')
plt.xlabel('Ciclo')
plt.legend(['entrenamiento', 'validación'], loc='lower right')
plt.show()
```



[64]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Pérdida del modelo')
```

```
plt.ylabel('Pérdida')
plt.xlabel('Ciclo')
plt.legend(['entrenamiento', 'validación'], loc='upper right')
plt.show()
```



[14]:
```
test_loss, test_acc = cnn.evaluate(x_test, y_test)
```

```
2/2 -------------- 2s 592ms/step -
accuracy: 0.8542 - loss: 1.0418
```

[16]:
```
test_loss, test_acc
```

[16]: (1.0474333763122559, 0.84375)

[24]:
```
from sklearn.metrics import accuracy_score, recall_score
```

[24]:
```
recall_score(Y_true, Y_pred_classes, average='macro', zero_division=np.nan )
```

[24]: 0.83245670995671

[25]:
```
accuracy_score(Y_true, Y_pred_classes)
```

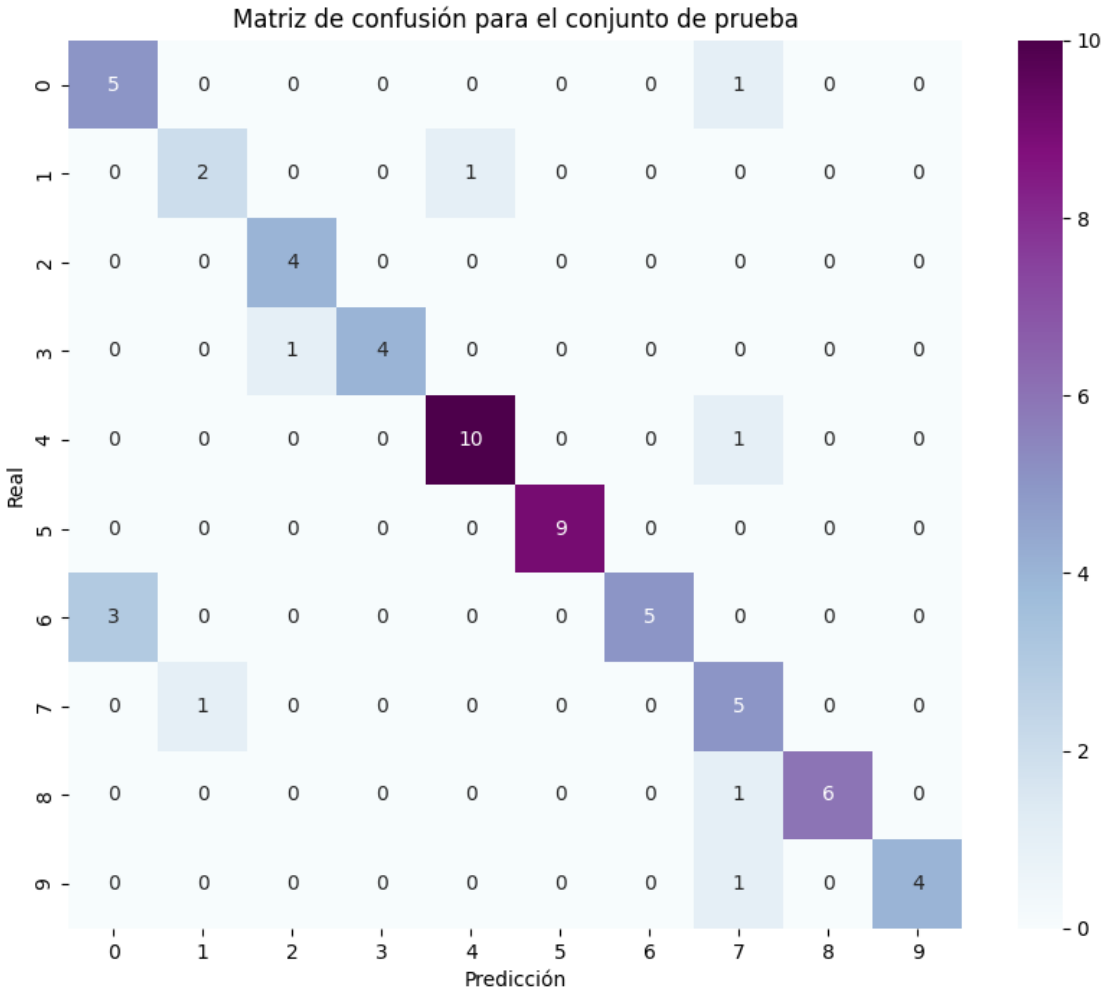
[25]: 0.84375

[19]:
```
# Predict the values from the testing dataset
Y_pred = cnn.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert testing observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = tensorflow.math.confusion_matrix(Y_true, Y_pred_classes)
```

```
2/2 -------------- 1s 505ms/step
```

[14]:
```
import seaborn as sns
```

[21]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, annot=True, fmt='g', cmap="BuPu")
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de confusión para el conjunto de prueba')
plt.show()
```



[7]:
```
test_datagen = ImageDataGenerator(
    rescale=1./255.
)
path_hoset = './drive/MyDrive/Entrenamiento/' + 'Hold_Out_Set/'
holdout_set = test_datagen.flow_from_directory(
    path_hoset,
    target_size=(128,128),
    class_mode='categorical',
    batch_size=779,
    color_mode='grayscale'
)
```

```
Found 779 images belonging to 10 classes.
```

[8]:
```
for x_batch, y_batch in holdout_set:
    x_holdout = x_batch
```

```
    y_holdout = y_batch
    break
```

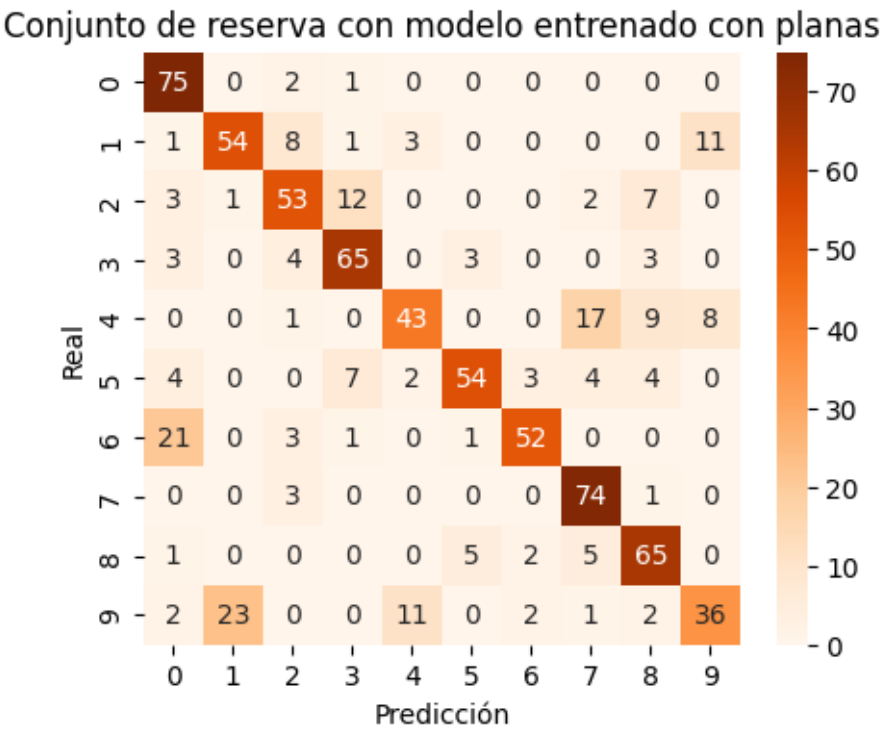
[10]:
```
model_path = './drive/MyDrive/Modelo/CNN_MNIST.keras'
modelo_MNIST = tensorflow.keras.models.load_model(model_path)
```

[11]:
```
model_path = './drive/MyDrive/Modelo/CNN01.keras'
cnn = tensorflow.keras.models.load_model(model_path)
```

[12]:
```
# Predict the values from the testing dataset
result_CNN = cnn.predict(x_holdout)
# Convert predictions classes to one hot vectors
Y_pred_classes_cnn = np.argmax(result_CNN,axis = 1)
# Convert testing observations to one hot vectors
Y_true_cnn = np.argmax(y_holdout,axis = 1)
# compute the confusion matrix
confusion_mtx_cnn = tensorflow.math.confusion_matrix(Y_true_cnn,␣
 ↪Y_pred_classes_cnn)
```

```
25/25 -------------- 21s 847ms/step
```

[21]:
```
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_mtx_cnn, annot=True, fmt='g', cmap="Oranges")
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Conjunto de reserva con modelo entrenado con planas')
plt.show()
```



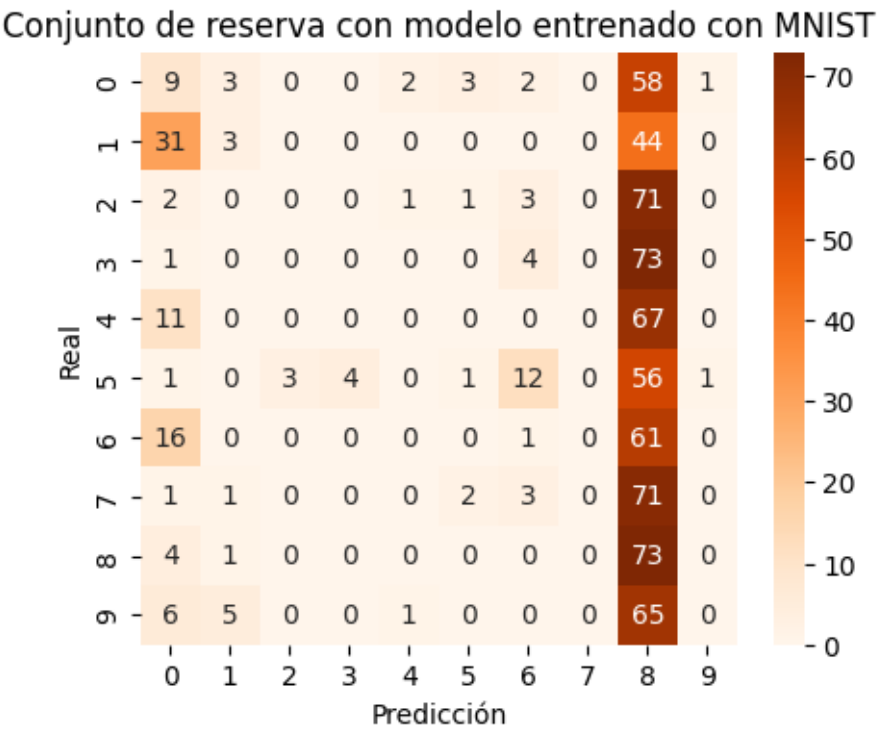
[25]:
```
accuracy_score(Y_true_cnn, Y_pred_classes_cnn)
```

[25]: 0.7329910141206675

[16]:
```
# Predict the values from the testing dataset
results_mnist = modelo_MNIST.predict(x_holdout)
# Convert predictions classes to one hot vectors
Y_pred_classes_mnist = np.argmax(results_mnist,axis = 1)
# Convert testing observations to one hot vectors
Y_true_mnist = np.argmax(y_holdout,axis = 1)
# compute the confusion matrix
confusion_mtx_mnist = tensorflow.math.confusion_matrix(Y_true_mnist,␣
 ↪Y_pred_classes_mnist)
```

```
25/25 ---------------- 35s 1s/step
```

[22]:
```
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_mtx_mnist, annot=True, fmt='g', cmap="Oranges")
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Conjunto de reserva con modelo entrenado con MNIST')
plt.show()
```



[26]:
```
accuracy_score(Y_true_mnist, Y_pred_classes_mnist)
```

[26]: 0.1116816431322208

C. Sistema de calificación

Código para la construcción del sistema de calificación, utilizando una de las tareas como ejemplo de su funcionamiento.

[ ]:
```
pip install pdf2image
```

```
Collecting pdf2image
  Downloading pdf2image-1.16.0-py3-none-any.whl (10 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages
(from pdf2image) (7.1.2)
Installing collected packages: pdf2image
Successfully installed pdf2image-1.16.0
```

[ ]:
```
!apt-get install poppler-utils
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  poppler-utils
0 upgraded, 1 newly installed, 0 to remove and 37 not upgraded.
Need to get 154 kB of archives.
After this operation, 613 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 poppler-utils
amd64 0.62.0-2ubuntu2.12 [154 kB]
Fetched 154 kB in 1s (203 kB/s)
Selecting previously unselected package poppler-utils.
(Reading database ... 155222 files and directories currently installed.)
Preparing to unpack .../poppler-utils_0.62.0-2ubuntu2.12_amd64.deb ...
Unpacking poppler-utils (0.62.0-2ubuntu2.12) ...
Setting up poppler-utils (0.62.0-2ubuntu2.12) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

[ ]:
```
from google.colab.patches import cv2_imshow
import cv2
import numpy as np
from pdf2image import convert_from_path
import matplotlib.pyplot as plt
from os import listdir, mkdir, path
import tensorflow
import sys
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
root = './drive/MyDrive/Tareas/'
path = './drive/MyDrive/Respuestas/Tarea-2/'
```

[ ]:
```
tarea = cv2.imread(path+ 'Tarea2_png/'  + 'Respuestas2_202109625.png')
```

[ ]:
```
import argparse
import imutils
def sort_contours(cnts, method="left-to-right"):
        # initialize the reverse flag and sort index
        reverse = False
        i = 0
        # handle if we need to sort in reverse
        if method == "right-to-left" or method == "bottom-to-top":
                reverse = True
        # handle if we are sorting against the y-coordinate rather than
        # the x-coordinate of the bounding box
        if method == "top-to-bottom" or method == "bottom-to-top":
                i = 1
        # construct the list of bounding boxes and sort them from top to
        # bottom
        boundingBoxes = [cv2.boundingRect(c) for c in cnts]
        (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
                key=lambda b:b[1][i], reverse=reverse))
        # return the list of sorted contours and bounding boxes
        return (cnts, boundingBoxes)
```

[ ]:
```
def crea_contornos(imagen):
  #width = 4250
  #height = 5500
  #dim = (width, height)

  #Leer la imagen
  img = cv2.imread(imagen)
  #Redimensionarla
  #resized = cv2.resize(img1, dim, interpolation = cv2.INTER_AREA)
  #cv2.imwrite(imagen,resized)
  #img = cv2.imread(imagen)

  #Mejora el contraste
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  pre_thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.
 ↪THRESH_BINARY, 21, 10)

  #Guarda una nueva imagen
  cv2.imwrite(imagen, pre_thresh)

  #Volvemos a leerla
  img = cv2.imread(imagen, 0)

  #Umbral de la imagen
  (thresh, img_bin) = cv2.threshold(img, 100, 255,cv2.THRESH_BINARY | cv2.
 ↪THRESH_OTSU)
```

```
 #Invertir la imagen
 img_bin = 255-img_bin
 cv2.imwrite("Image_bin.jpg",img_bin) # DEBUG

 # Definimos un kernel del tamaño de la imagen
 kernel_length = np.array(img).shape[1]//80
#Un kernel vertical (1 x longitud del kernel) que detecta todas las líneas␣
↪verticales de la imagen.
 verticle_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, kernel_length))
 # Un kernel horizontal (longitud del kernel x 1) que detecta todas las líneas␣
↪horizontales de la imagen.
 hori_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_length, 1))
 # A kernel of (3 X 3) ones.
 kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

 # Operación morfológica (erosión) para detectar líneas verticales de una␣
↪imagen.
 img_temp1 = cv2.erode(img_bin, verticle_kernel, iterations=3)
 verticle_lines_img = cv2.dilate(img_temp1, verticle_kernel, iterations=15)
 cv2.imwrite("verticle_lines.jpg",verticle_lines_img) # DEBUG
 # Operación morfológica (erosión) para detectar líneas horizontales de una␣
↪imagen.
 img_temp2 = cv2.erode(img_bin, hori_kernel, iterations=3)
 horizontal_lines_img = cv2.dilate(img_temp2, hori_kernel, iterations=15)
 cv2.imwrite("horizontal_lines.jpg",horizontal_lines_img) # DEBUG

 # Parámetros de ponderación, esto decidirá la cantidad de imagen que se␣
↪agregará para crear una nueva imagen.
 alpha = 0.5
 beta = 1.0 - alpha
 # Esta función ayuda a agregar dos imágenes con un parámetro de peso␣
↪específico para obtener una tercera imagen como suma de dos imágenes.
 img_final_bin = cv2.addWeighted(verticle_lines_img, alpha,␣
↪horizontal_lines_img, beta, 0.0)
 img_final_bin = cv2.erode(~img_final_bin, kernel, iterations=2)
 (thresh, img_final_bin) = cv2.threshold(img_final_bin, 128,255, cv2.
↪THRESH_BINARY | cv2.THRESH_OTSU)
 cv2.imwrite("img_final_bin.jpg",img_final_bin)  # DEBUG

 # Encuentra contornos para la imagen, que detectará todos los recuadros.
 contours, hierarchy = cv2.findContours(
 img_final_bin, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
 # Ordena todos los contornos de arriba a abajo.
 (contours, boundingBoxes) = sort_contours(contours, method="top-to-bottom")

 return contours, img
```

[ ]:
```
contornos, imagen = crea_contornos(root+ 'Tarea2_png/'  + 'Respuestas2_202109625.
↪png')
```

[ ]:
```
def limpia_contornos(contours):
  idx = 0
  imagenes= {}
  for c in contours:
    x, y, w, h = cv2.boundingRect(c)
    if w*h > 60000 and w*h < 90000 and x != 0:
      idx += 1
      coor_list = [0,0,0,0]
      coor_list[0] = y
      coor_list[1] = y+h
      coor_list[2] = x
      coor_list[3] = x+w
      imagenes[str(idx)] = coor_list
  imagenes

  id_sinborde = set(range(1,len(imagenes)+1))
  for i in range(1,len(imagenes)+1):
    for j in range(1,len(imagenes)+1) :
      if imagenes[str(i)][0] <= imagenes[str(j)][0] and imagenes[str(i)][1] >␣
↪imagenes[str(j)][1] and imagenes[str(i)][2] <= imagenes[str(j)][2] and␣
↪imagenes[str(i)][3] > imagenes[str(j)][3]:
        #print(i, 'se va')
        id_sinborde = id_sinborde - set([i,i])

  return imagenes,id_sinborde
```

[ ]:
```
imagenes, sinborde = limpia_contornos(contornos)
```

[ ]:
```
imagenes
```

[ ]:
```
{'1': [2745, 3016, 1676, 1940],
 '10': [3654, 3924, 1024, 1285],
 '11': [3654, 3924, 720, 984],
 '2': [2745, 3016, 1374, 1638],
 '3': [2745, 3016, 722, 984],
 '4': [3198, 3469, 1676, 1940],
 '5': [3198, 3469, 1374, 1638],
 '6': [3198, 3469, 1024, 1285],
 '7': [3198, 3469, 720, 984],
 '8': [3654, 3924, 1676, 1940],
 '9': [3654, 3924, 1374, 1638]}
```

[ ]:
```
imagenes['12'] = [2745, 3016, 1024, 1285]
```

[ ]:
```
sinborde.add(12)
```

[ ]:
```
def etiqueta_imagenes(imagenes,id_sinborde):

  imagenes_bak = {}
  for i in id_sinborde:
    imagenes_bak[str(i)] = imagenes[str(i)]
  fila = 0
  etiquetas_final = {}
  while len(imagenes_bak)>0:
    #calculamos las areas anteriores a la coordenada inicial
    fila += 1
    area_anterior = {}
    for k in imagenes_bak.keys():
      area_anterior_k = imagenes_bak[k][0]*imagenes_bak[k][2]
      area_anterior[k]=area_anterior_k

    #encontramos el area minima de dichas areas
    min_area_anterior = np.min(np.array([area_anterior[k] for k in area_anterior.
↪keys()]))

    #encontramos la llave del area minima
    for k in area_anterior.keys():
      if area_anterior[k]==min_area_anterior:
        primer_elemento_fila = k

    y_min = imagenes_bak[primer_elemento_fila][0] - 0.
↪2*(imagenes_bak[primer_elemento_fila][1]-imagenes_bak[primer_elemento_fila][0])
    y_max = imagenes_bak[primer_elemento_fila][0] + 0.
↪2*(imagenes_bak[primer_elemento_fila][1]-imagenes_bak[primer_elemento_fila][0])

    fila_k = []
    for k in imagenes_bak.keys():
      if imagenes_bak[k][0] > y_min and y_max > imagenes_bak[k][0]:
        #print(k,'esta en la misma fila que ',primer_elemento_fila)
        fila_k.append(k)

    posiciones_fila = {}
    contador_inciso = 0
    for pos_x in np.sort(np.array([imagenes_bak[k][2] for k in fila_k])):
      contador_inciso += 1
      for k in fila_k:
        if imagenes_bak[k][2] == pos_x:
          posiciones_fila[k] = contador_inciso
          break

    etiquetas_fila = {}
```

```
    for k in posiciones_fila.keys():
      etiquetas_fila[k] = 'Inciso' + str(fila) + '_' + str(posiciones_fila[k])

    for k in posiciones_fila.keys():
      imagenes_bak.pop(k, None)

    for k in posiciones_fila.keys():
      etiquetas_final[k] = etiquetas_fila[k]

  return imagenes,etiquetas_final
```

[ ]:
```
imgs, etiquetas= etiqueta_imagenes(imagenes, sinborde)
```

[ ]:
```
imgs
```

[ ]:
```
{'1': [2745, 3016, 1676, 1940],
 '10': [3654, 3924, 1024, 1285],
 '11': [3654, 3924, 720, 984],
 '12': [2745, 3016, 1024, 1285],
 '2': [2745, 3016, 1374, 1638],
 '3': [2745, 3016, 722, 984],
 '4': [3198, 3469, 1676, 1940],
 '5': [3198, 3469, 1374, 1638],
 '6': [3198, 3469, 1024, 1285],
 '7': [3198, 3469, 720, 984],
 '8': [3654, 3924, 1676, 1940],
 '9': [3654, 3924, 1374, 1638]}
```

[ ]:
```
etiquetas
```

[ ]:
```
{'1': 'Inciso1_4',
 '10': 'Inciso3_2',
 '11': 'Inciso3_1',
 '12': 'Inciso1_2',
 '2': 'Inciso1_3',
 '3': 'Inciso1_1',
 '4': 'Inciso2_4',
 '5': 'Inciso2_3',
 '6': 'Inciso2_2',
 '7': 'Inciso2_1',
 '8': 'Inciso3_4',
 '9': 'Inciso3_3'}
```

[ ]:
```
def recuadros(imagenes, etiquetas_final):
  #contours, img = crea_contornos(imagen)
  #imagenes,id_sinborde = limpia_contornos(contours)
  #imagenes,etiquetas_final = etiqueta_imagenes(imagenes,id_sinborde)
```

```
  coor_imagenes = { }
  for key in etiquetas_final.keys():
    coor_imagenes[str(etiquetas_final[key])] = imagenes[str(key)]
    y1 = imagenes[str(key)][0]
    y2 = imagenes[str(key)][1]
    x1 = imagenes[str(key)][2]
    x2 = imagenes[str(key)][3]
    #new_img = img[y1:y2, x1:x2]
    #print(ruta+str(i) + '.png')
  #  cv2.imwrite(ruta+str(etiquetas_final[key]) + '.png', new_img)

  return coor_imagenes
```

[ ]:
```
coordenadas= recuadros(imgs, etiquetas)
```

[ ]:
```
coordenadas
```

[ ]:
```
{'Inciso1_1': [2745, 3016, 722, 984],
 'Inciso1_2': [2745, 3016, 1024, 1285],
 'Inciso1_3': [2745, 3016, 1374, 1638],
 'Inciso1_4': [2745, 3016, 1676, 1940],
 'Inciso2_1': [3198, 3469, 720, 984],
 'Inciso2_2': [3198, 3469, 1024, 1285],
 'Inciso2_3': [3198, 3469, 1374, 1638],
 'Inciso2_4': [3198, 3469, 1676, 1940],
 'Inciso3_1': [3654, 3924, 720, 984],
 'Inciso3_2': [3654, 3924, 1024, 1285],
 'Inciso3_3': [3654, 3924, 1374, 1638],
 'Inciso3_4': [3654, 3924, 1676, 1940]}
```

[ ]:
```
ruta = './drive/MyDrive/Respuestas/Tarea-2/202109625/'
for key in etiquetas.keys():
    y1 = imagenes[str(key)][0]
    y2 = imagenes[str(key)][1]
    x1 = imagenes[str(key)][2]
    x2 = imagenes[str(key)][3]
    new_img = imagen[y1:y2, x1:x2]
    #print(ruta+str(i) + '.png')
    cv2.imwrite(ruta+str(etiquetas[key]) + '.png', new_img)
```

[ ]:
```
#Esta no
def revisar(ruta_tarea, filas,incisos):
  respuestas = listdir(ruta_tarea)
  if len(respuestas) == filas*incisos:
    lista_resp = [ ]
    for respuesta in respuestas:
      respuesta1 = respuesta.replace('Inciso', '')
```

```
      respuesta2= respuesta1.replace('.png', '')
      coordenadas = respuesta2.split('_')
      coordenadas = list(map(int, coordenadas))
      lista_resp.append(coordenadas)
    matriz = np.array(lista_resp)
    max_f,max_c = np.max(matriz,axis=0)
    if max_f == filas and max_c == incisos:
      return True
    else:
      print(ruta_tarea, 'revisar')
      return False
  else:
    print(ruta_tarea, 'revisar')
    return False
```

[ ]:
```
def prediccion(ruta,model, predicciones, width = 128,  height = 128):
  dim = (width, height)
  predicciones['ruta']=ruta
  originales= {}
  for i in (listdir(ruta)):
    img = cv2.imread(ruta+ i,0)
    resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
    #plt.imshow(resized, cmap='gray')
    i=i.replace('.png','')
    originales[str(i)] =resized

  keys = list(originales.keys())
  predichos= { }
  for k in keys:
    input = np.resize(originales[str(k)],(1,128,128,1))
    predicho= model.predict(input)
    predichos[str(k)] = np.argmax(predicho)
  predicciones['predicciones'] = predichos
  return predicciones
```

[ ]:
```
def evaluacion(clave, predicciones):
  predicciones['evaluación'] = {}
  for key in predicciones['predicciones'].keys():
    respuesta = key
    respuesta1 = respuesta.replace('Inciso', '')
    respuesta2= respuesta1.replace('.png', '')
    coordenadas = respuesta2.split('_')
    coordenadas = list(map(int, coordenadas))
    respuesta_clave = clave[coordenadas[0]-1,coordenadas[1]-1]
    if predicciones['predicciones'][key] == respuesta_clave:
      predicciones['evaluación'][key]=1
    else:
```

```
      predicciones['evaluación'][key]=0
  return predicciones
```

[ ]:
```
def calificar(n,predicciones, path_respuestas):
  image = cv2.imread(predicciones['dir_original'])
  for key in predicciones['coordenadas'].keys():
    start_point = ( predicciones['coordenadas'][key][2],  ␣
 ↪predicciones['coordenadas'][key][0] )
    end_point = ( predicciones['coordenadas'][key][3],  ␣
 ↪predicciones['coordenadas'][key][1] )
    if predicciones['evaluación'][key] == 1:
      color = (0, 255, 0) #BGR
    else:
      color = (0, 0, 255) #BGR
    thickness = 4
    image = cv2.rectangle(image, start_point, end_point, color, thickness)
  cv2.imwrite(path_respuestas+ 'calificado' + str(n) + '.png', image)
```

[ ]:
```
model_path = './drive/MyDrive/Modelo/CNN01.h5'
new_model = tensorflow.keras.models.load_model(model_path)
diccionario_maestro = {}
dir = './drive/MyDrive/Respuestas/Tarea-2/202109625/'
root = './drive/MyDrive/Tareas/'
diccionario_maestro['202109625'] = {}
diccionario_maestro['202109625']['dir_original'] = root + 'Tarea2_png/
 ↪Respuestas2_202109625.png'
diccionario_maestro['202109625']['coordenadas'] = coordenadas
diccionario_maestro['202109625']= prediccion(dir, new_model,␣
 ↪diccionario_maestro['202109625'] )
diccionario_maestro['202109625']= evaluacion( np.
 ↪array([[0,1,5,7],[0,2,2,3],[0,1,0,0]]), diccionario_maestro['202109625'])
calificar('202109625',diccionario_maestro['202109625'], './drive/MyDrive/
 ↪Calificaciones/Tarea2/')
```