
Universidad Nacional de Rosario

Tesina de grado
para la obtención del grado de
Licenciado en Ciencias de la Computación

# Clasificación de variedades de cultivo de sorgo utilizando redes convolucionales profundas

*Autor:*
Bruno A. Baruffaldi

*Director:*
Dr. Diego S. Pérez

*Co-director:*
Dr. Flavio E. Spetale

Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, Santa Fe, Argentina

# Agradecimientos

Esta tesina representa el trabajo y acompañamiento de mucha gente a lo largo de estos años. Por eso, quiero expresar mi profundo agradecimiento a las siguientes personas:

- A mi familia por su apoyo incondicional y por creer en mí.
- A mi director y co director, Dr. Diego Perez y Dr. Flavio Spetale, por su paciencia, su guía y su constante motivación. Su conocimiento y experiencia han sido de gran ayuda en el desarrollo de mi investigación.
- A mis profesores, por su valiosa contribución a lo largo de la carrera y por compartir sus conocimientos y habilidades conmigo.
- A toda aquella persona que forma parte, de una forma u otra, de esta gran comunidad de LCC.

A todos los mencionados y los que seguramente falte mencionar, les agradezco de corazón por todo lo que recibí.

# Índice general

# 1 Introducción

## 1.1. Resumen

El creciente aumento demográfico a nivel mundial está provocando la necesidad de incrementar la producción agrícola. Científicos y productores buscan obtener variedades genéticas más productivas, resistentes y menos exigentes respecto al consumo de recursos. Una de las herramientas fundamentales para este proceso es el fenotipado de plantas que ofrece soluciones para una gran variedad de problemas agronómicos como ser clasificación de especies y variedades o evaluación del crecimiento de plantas, entre otras. Sin embargo, las herramientas tradicionales de fenotipado se basan en mediciones manuales de rasgos seleccionados, atributos, en una pequeña muestra de plantas y, por tanto, impiden de un análisis completo de todos sus rasgos dentro de una sola planta y entre cultivares (especies). Esto limita la capacidad de comprender cómo los fenotipos expresados se correlacionan con factores genéticos subyacentes y condiciones ambientales; retrasando el avance científico en problemas relacionados con la reproducción como la resistencia a la sequía. Por otro lado, las técnicas basadas en imágenes tienen potencial para aumentar enormemente la escala y el rendimiento de las actividades de fenotipado de plantas. Estas herramientas computacionales deben transformar automáticamente las imágenes en mediciones fenotípicas confiables y precisas. Las técnicas de aprendizaje automático, y en particular el aprendizaje profundo, tienen la virtud de mejorar la solidez del fenotipado basado en imágenes y extenderse hacia características fenotípicas más complejas y abstractas. Las redes neuronales convolucionales profundas (CNN) son métodos de aprendizaje profundo que se adaptan particularmente bien a los problemas de visión por computadora. A diferencia de los enfoques clásicos de visión por computadora que primero miden las propiedades estadísticas de la imagen como características para aprender un modelo de los datos, las CNN aprenden activamente una variedad de patrones visuales durante el entrenamiento del modelo. Desafortunadamente, las CNN pueden demandar una gran cantidad de cómputo durante su ejecución por lo que podrían no ser efectivas como solución en situaciones que requieren procesamiento en tiempo real con hardware limitado. Esto motivo en los últimos años al desarrollo de estrategias rigurosas de fitomejoramiento para seleccionar rasgos que sean valorados para cada fin. En particular, en esta tesina se utilizará el sorgo como cultivo ya que es un grano sin gluten que puede ser utilizado como sustituto de alimentos agrícolas y fuente de biocombustible. En base a

esto, se propone el desarrollo e implementación de una nueva metodología de aprendizaje automatizado que usa una arquitectura convolucional profunda basada en ResNet para la clasificación de especies de sorgo con el objeto de mejorar el tiempo de ejecución y la latencia para que sea implementable en sobre sistemas embebidos comúnmente utilizados en entornos industriales. Para llevar a cabo esta tarea, se propone modificar la arquitectura e incorporar las convoluciones BSC con el objeto de reducir el costo computacional de las CNN. Asimismo, para contrarrestar la degradación en la precisión del método introducida por las convoluciones BSC se propone utilizar función de costos MCLoss, que permite especializar los mapas de características de una capa intermedia de la red en clases especificas.

## 1.2. Organización del trabajo

Esta tesina esta organizada en 6 capítulos. El Capítulo 1 consta del resumen de la tesina; el Capítulo 2 detalla los conceptos relacionados al Aprendizaje Automatizado a fines a este tesina. El Capítulo 3 describe trabajos relacionados con esta tesina que nos ayudan a comprender mejor la problemática abordada. El Capítulo 4 detalla la metodología desarrollada para la predicción automática de variedades de sorgo por medio de redes neuronales profundas. El Capítulo 5 muestra los resultados obtenidos con nuestra metodología y compara su rendimiento contra los sistemas embebidos Jetson Xavier NX y Jetson Nano. Por último, el Capítulo 6 presenta el resumen final de la tesina y los posibles proyectos a futuro.

# 2 Marco Teórico

## 2.1. Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo

En esta sección se explican los conceptos de Inteligencia Artificial(IA), Aprendizaje Automático (AA) y Aprendizaje Profundo (AP) y como se relacionan (Figura 1).

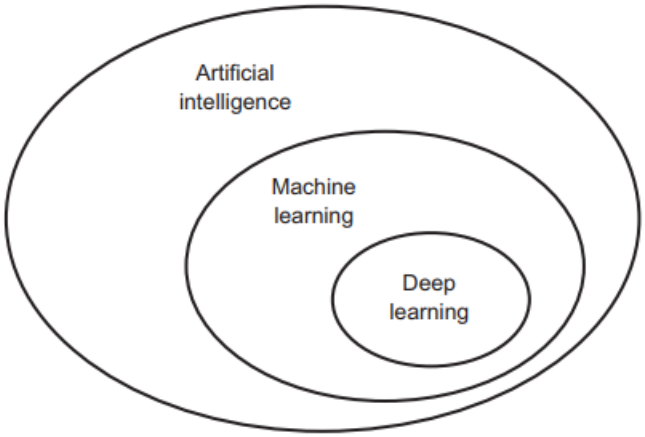



Figura 1: Diagrama de Venn con los conceptos: Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo [1].

### Inteligencia Artificial (IA)

El término Inteligencia Artificial fue enunciado en 1956 por Allen Newell, Herbert Simon, John McCarthy, Marvin Minsky y Arthur Samuel, considerados hoy como fundadores y primeros líderes de la investigación en este campo.

Una definición concisa sería: *el esfuerzo por automatizar tareas intelectuales normalmente realizadas por humanos*. Como tal, la IA es un campo general que engloba el aprendizaje automático y el aprendizaje profundo pero que también incluye muchos más enfoques que no implican ningún aprendizaje como es el caso de las heurísticas [1].

Durante mucho tiempo, los expertos creyeron que la IA a nivel humano podía lograrse haciendo que los programadores escribieran a mano un conjunto suficientemente amplio de reglas explícitas para manipular el conocimiento. A este enfoque se lo conoce como IA simbólica y fue el paradigma dominante entre los años 50 y los finales de los 80 donde alcanzó su máxima popularidad durante el auge de los sistemas expertos en la década de 1980. Sin embargo, estos sistemas eran muy efectivos en resolver problemas bien definidos y lógicos, como por ejemplo jugar al ajedrez, pero resultaban intratables cuando había que definir reglas para solucionar problemas complejos como visión por computadora, reconocimiento de voz o traducción automática. Por tanto, surgió un nuevo enfoque: el *Aprendizaje Automático* [1].

## Aprendizaje Automatizado (AA)

El aprendizaje automático surge de las preguntas: *¿Podría una computadora ir más allá de lo que sabemos ordenarle que haga y aprender por sí sola a realizar una tarea determinada?, y ¿Podría un ordenador sorprendernos?*

Estas preguntas dieron lugar a un nuevo paradigma de programación conocido como *Aprendizaje Automatizado* (Figura 2). En la programación clásica, los humanos escriben un conjunto de reglas, un programa, y proveen datos para ser procesados mediante las mismas obteniendo las respuestas de este procesamiento. En el AA, en cambio, se proveen los datos y las respuestas esperadas de estos datos a una serie de algoritmos particulares de AA y se obtienen reglas que luego pueden ser aplicadas a nuevos ejemplos para obtener respuestas desconocidas [1].

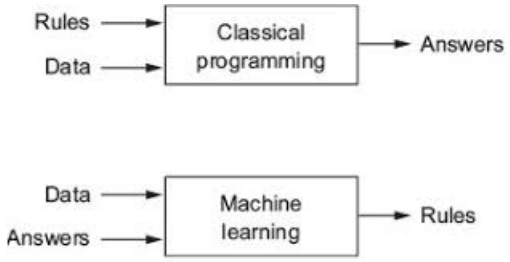



Figura 2: Diferencia entre Aprendizaje Automatizado y Programación clásica. [1]

Tom M. Mitchell [2] definió el *Aprendizaje Automatizado* como: *"Un al-*

*goritmo aprende de la experiencia E, con respecto a una tarea T y medida de rendimiento P, si su rendimiento al realizar la tarea T (medido por P) aumenta con la experiencia E"*. Por esto se dice que un sistema de AA es entrenado, en vez de programado. Durante este proceso de entrenamiento se proporcionan al sistema muchos ejemplos relevantes para una tarea específica, y este encuentra patrones estadísticos en los datos que le permiten generar un modelo para automatizar la tarea. Este modelo "capacita" al sistema no solo para resolver la tarea en los ejemplos vistos, sino también para generalizar y funcionar en datos no vistos durante el proceso de entrenamiento. Usualmente, al utilizar estos sistemas se suelen dividir a los conjuntos de datos en dos subgrupos disjuntos, uno utilizado para llevar a cabo el proceso de entrenamiento, conocido como conjunto de entrenamiento, y el otro destinado a evaluar el funcionamiento del sistema sobre ejemplos no vistos, conocido como conjunto de prueba.

El AA empezó a florecer hasta la década de 1990 y rápidamente se convirtió en el subcampo más popular y exitoso de la IA impulsada por las mejoras en el hardware y la disponibilidad de conjuntos de datos más grandes. El AA sienta sus bases en la matemática y la estadística pero debemos tener en cuenta que debido a la complejidad de los datos, conjuntos de millones de imágenes que consisten en decenas de miles de píxeles cada una, los análisis estadísticos clásicos resultan poco prácticos. Por lo tanto, el Aprendizaje Automático - y más aún el Aprendizaje Profundo - son disciplinas prácticas donde las ideas se prueban empíricamente con mayor frecuencia que de manera teórica [1].

## Aprendizaje Profundo (AP)

El aprendizaje profundo es un subcampo específico del aprendizaje automatizado. El AP es una nueva forma de aprender representaciones a partir de datos haciendo hincapié en el aprendizaje de capas sucesivas de representaciones cada vez más abstractas. El aprendizaje profundo moderno a menudo implica decenas o incluso cientos de capas sucesivas de representaciones que se aprenden de manera conjunta, cada una siendo modificada para cumplir con las necesidades de representación de la capa superior como así también las de la capa inferior, como se muestra en Figura 3 [1].

El AP revolucionó el campo del Aprendizaje Automático con resultados impresionantes en problemas de percepción, como la vista y la audición, que resultan naturales e intuitivos para los humanos pero fueron siempre un problema de gran complejidad para las computadoras.

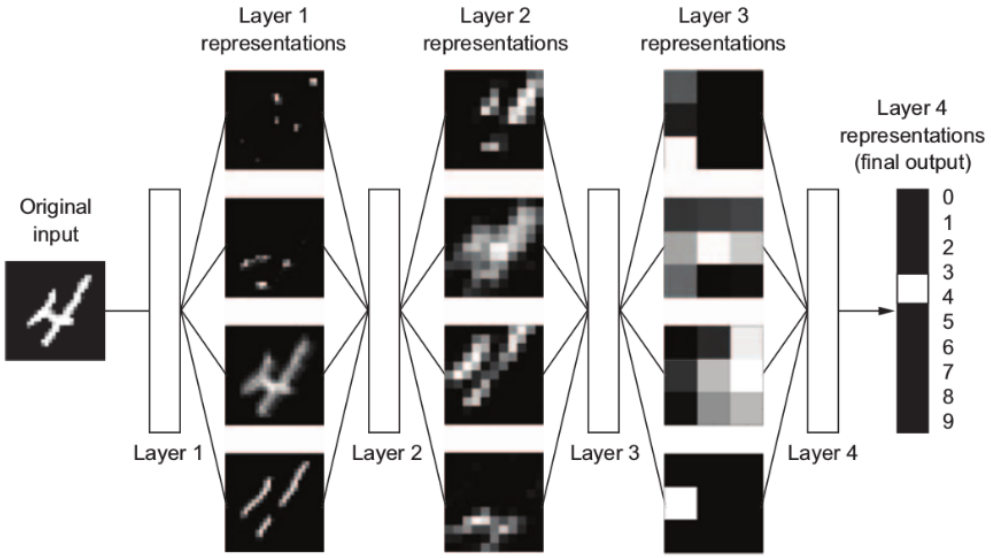



Figura 3: Arquitectura de una red neuronal profunda para la clasificación de dígitos [1].

## 2.2. Redes Neuronales Artificiales

Las redes neuronales artificiales (de ahora en más, redes neuronales o RNA) son una familia de modelos que proveen un enfoque robusto para aproximar funciones a valores reales, discretos y vectoriales [3].

Las redes neuronales están compuestas por una colección de unidades conectadas, llamadas neuronas artificiales o *perceptrones* (Figura 4).

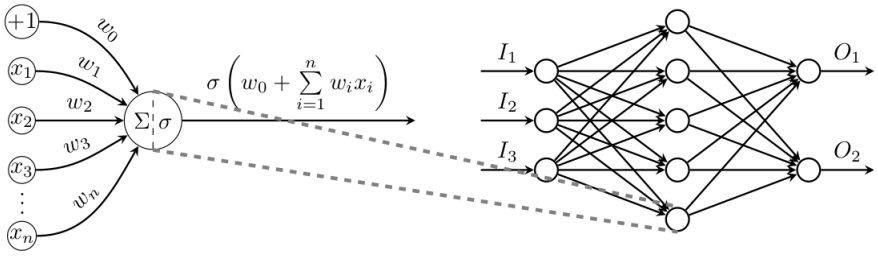



Figura 4: Neurona o perceptrón de una RNA. La información fluye de izquierda a derecha.

Las conexiones entre neuronas tienen normalmente asociado un peso $w_i$

que se ajusta en el período de entrenamiento. Formalmente una neurona modela la función:

$$y = \sigma(w_0 + \sum_{i=1}^{n} w_i * x_i),$$

donde $x_i$ es un valor de entrada conectado a través de una conexión con peso $w_i$, $n$ es el número de conexiones, $w_0$ es un peso inicial, y $\sigma$ es la función de activación, usualmente una de las siguientes:

- sigmoidea: $\sigma(x) = \frac{1}{1+e^{-x}}$
- tangente hiperbólica: $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- lineal rectificada: $\sigma(x) = \begin{cases} 0, & \text{si } x \leq 0 \\ x, & \text{caso contrario} \end{cases}$

Esta última función de activación es la utilizada en esta tesina, donde las unidades que emplean el rectificador son conocidas como $ReLU$ [4] (del inglés Rectified Linear Unit).

Entrenar una red artificial implica obtener un vector de pesos $\vec{v} = (w_0, w_1, \ldots, w_n)$ para cada neurona de la red, lo que resulta en un conjunto total de parámetros a entrenar $H = (\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_m)$ siendo $m$ la cantidad total de neuronas. Las capas donde cada una de sus neuronas se conecta con todas las neuronas la capa anterior son denominadas *fully connected*. En términos generales las redes neuronales cuentan con 3 tipos de capas. La capa de entrada almacenaran los datos a procesar, la capa de salida guardará los resultados del modelo y las capas intermedias. Estas capas intermedias serán procesadas de manera sucesiva, de forma tal que los valores de entrada de la capa $N$ son los valores de salida de la capa $N-1$. En el caso de la Figura 3, la capa 1 es la capa de entrada, la capa 4 es la de salida y el resto son las capas intermedias.

El proceso de entrenamiento de las redes neuronales suele ser más costoso que el entrenamiento de otros modelos de AA. Durante este proceso, la red actualiza poco a poco los valores de sus pesos para mejorar su desempeño en la tarea específica, lo que implica examinar varias veces el conjunto de datos. La duración del entrenamiento está determinada por la cantidad de veces (o épocas) que la red procesa el total del conjunto de datos, y el impacto de estas actualizaciones depende de una serie de parámetros (como el learning rate, momentum y el decaimiento de pesos). Para reducir el costo de este proceso, se puede utilizar el pre-entrenamiento de modelos, una técnica de

transferencia de aprendizaje que implica entrenar un modelo en una tarea relacionada antes de entrenarlo definitivamente para la tarea específica. La idea detrás del pre-entrenamiento es que el modelo aprenda características generales del dominio en el que se está trabajando, lo que puede ayudar a mejorar su rendimiento en tareas específicas y reducir la cantidad de épocas necesarias para obtener resultados satisfactorios.

Las redes neuronales que utilizan perceptrones tienen una limitación clave: la complejidad computacional requerida para trabajar con imágenes. Los conjuntos de datos de imágenes simples, como la base de datos MNIST[5] de dígitos escritos a mano, son manejables para la mayoría de las arquitecturas basadas en perceptrones debido a su baja dimensionalidad de imagen de solo 28×28. En este conjunto de datos, una neurona en la primera capa contendrá 784(28x28x1) parámetros. Sin embargo, para imágenes en color de mayor tamaño, como 64x64, la cantidad de parámetros en una sola neurona de la primera capa aumenta significativamente a 12.288 (64x64x3). Para abordar esta problemática, surgieron las redes convolucionales que utilizan la localidad espacial de una imagen para reducir la cantidad de parámetros necesarios y trabajar de manera más eficiente sobre este tipo de problemas[6].

## 2.3. Redes Convolucionales

Las redes neuronales convolucionales, o CNNs (del inglés Convolutional Neural Networks), fueron propuestas por Yann LeCun[7] en los años '90 y relegadas por la comunidad debido a su alto requerimiento computacional. En 2012, volvieron a tomar relevancia cuando Krizhevsky[8] gana la competencia 2012 ImageNet Challenge por amplio margen y desde ese entonces ocupan un lugar principal en el estado del arte de *Computer Vision*.

Una CNN es una red neuronal con pesos compartidos y conexiones locales, donde cada neurona de salida de la capa convolucional se conecta con un subconjunto de neuronas en la entrada de manera regular y compartiendo los pesos $w$.

La aplicación de una capa convolucional sobre una matriz puede ser definida similarmente a una convolución (muchas librerías de aprendizaje automatizado llaman a la operación convolución pero matemáticamente se denomina cross-correlation). La convolución puede ser definida como:

$$S(i,j) = (I * K)(i,j) = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} I(i+a, j+b) \cdot K(a,b) \tag{1}$$

donde $I$ representa la matriz de entrada, $K$ es una matriz de $k \times k$ elementos llamada el *kernel* de la convolución y la salida $S$ es usualmente referida como un *feature map*. La Figura 5 muestra un ejemplo del cálculo de una convolución considerando una matriz $I$ de 7x7, una matriz $k$ de 3x3 y tomando $i = 0$ y $j = 3$; se obtiene $S(0,3) = (1*1+0*0+0*1)+(1*0+1*1+0*0)+(1*1+1*0+1*1) = 4$

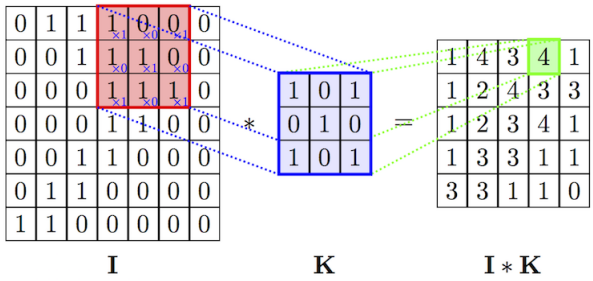



Figura 5: Ejemplo de una convolución 2D donde I representa los datos de entrada, K el kernel de la convolución aplicada y I*K (S) el resultado denominada *feature map*.

Para cada capa convolucional, los valores de los pesos de su *kernel* son los que se obtienen durante la etapa de entrenamiento, que intuitivamente funcionan como filtros de ciertas características relevantes en la imagen, como pueden ser aristas y esquinas en capas cercanas a la entrada, y formas complejas en capas posteriores. Las CNN son especialmente útiles para reconocer patrones en imágenes porque poseen invarianza traslacional (una característica sera reconocida independientemente de la zona de la imagen donde se encuentre).

Se puede hacer una distinción entre tres tipos de capas utilizadas en las CNNs:

- Capas Convolucionales: Se encargan de propagar a través de toda su entrada el kernel antes mencionado. La única salvedad es que se define un paso de aplicación, de esta manera se "saltean" algunas posiciones del kernel lo que reduce el costo computacional (Figura 5).

- Capas de Pooling: Una operación de pooling resume la salida de la capa anterior combinando sus valores utilizando diferentes funciones estadísticas. Por ejemplo, max pooling, reporta el valor máximo de un área rectangular de la entrada. Otras operación de pooling pueden ser el promedio de un entorno rectangular o un promedio ponderado basado en distancia al píxel central. Las capas de pooling reducen la dimensionalidad de la red aumentando la eficiencia y ayuda a controlar el sobre ajuste (Figura 6).

- Capas de Normalización: Existen distintas técnicas de pre-procesamiento que permiten estandarizar los datos con los que se va a trabajar. La más común es la normalización por lotes, uno de los métodos mas populares, es un tipo de capa introducida en 2015 por Ioffe y Szegedy[9] que permite normalizar datos dentro de las capas ocultas de la red de forma adaptativa incluso cuando la media y la varianza cambian con el tiempo durante el entrenamiento. Funciona manteniendo internamente una media móvil exponencial de la media y la varianza por lotes de los datos observados durante el entrenamiento. La normalización por lotes principalmente ayuda a la propagación del gradiente y, por tanto, permite crear redes más profundas [10].

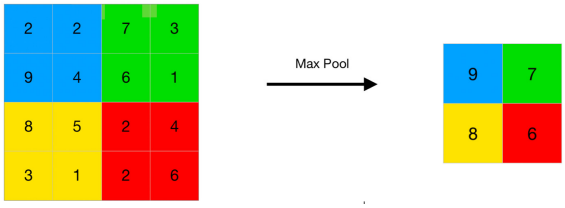



Figura 6: Ejemplo de operación *max pooling*. Se produce una reducción de la matriz tomando el valor máximo de cada región que se encuentra representada por un color especifico.

Usualmente una capa conceptual de las CNNs, que se repite varias veces, involucra 3 etapas: una primera capa de convolución, seguida por una función de activación (ReLU, por ejemplo) y finalmente una capa de pooling.

## Capas convolucionales

En las CNN tradicionales, una operación de convolución suele aplicar un conjunto de filtros a un grupo de matrices de entrada, produciendo un

*feature map* de salida, también conocido como mapa de activación o mapa de características. Esta operación puede ser costosa desde el punto de vista computacional, especialmente cuando el *feature map* de entrada tiene un gran tamaño espacial o el número de filtros es elevado. En las aplicaciones prácticas, la capacidad de cálculo es limitada y sobre todo en el contexto de sistemas embebidos. Este hecho dio lugar a otra importante línea de investigación centrada en mejorar la eficiencia de los modelos. Dentro de esta línea, con el objetivo de simplificar esta operación se desarrollaron nuevas variaciones de las convoluciones estándar. A continuación, se detalla la variante Blueprint Separable Convolution utilizada en esta tesina.

**Blueprint Separable Convolution**

En las CNN convencionales, se puede pensar cada capa convolucional como una operación que transforma una matriz de entrada $U$ de tamaño $M \times Y \times X$ en una matriz de salida $V$ de tamaño $N \times Y' \times X'$ aplicando filtros de convolución $F_1, ..., F_N$ , cada uno de tamaño $M \times K \times K$ tal que

$$V_n = U \cdot F_n \tag{2}$$

con $n \in 1, ..., N$. Los valores de los pesos de estos *kernels* se optimizan durante la fase de entrenamiento de las CNN y pueden tomar valores arbitrarios. Sin embargo, en [11] se muestra que existen correlaciones entre los distintos kernels de un mismo filtro. En Figura 7 se muestra un ejemplo de estas correlaciones, amplificando 3 kernels para resaltar el patrón que se repite en muchos de ellos. Estas correlaciones permiten aproximar a un filtro $F_n$ de dimensiones $M \times K \times K$ mediante un único *kernel* de dimensiones $K \times K$ (al que llamaremos "patrón") y $M$ escalares. Concretamente, se define que cada filtro $F_n$ se representa mediante un "patrón" $B_n$ y los escalares $\vec{w}_n = (w_{n,1}, ..., w_{n,M}) \in \mathbb{R}$

$$F_n = \vec{w}_n \cdot B_n \tag{3}$$

con $n \in 1, ..., N$. Este hallazgo es la base de las convoluciones separables por patrones (BSC) y aunque esta definición impone una dura restricción a los valores que pueden tomar los pesos de un *kernel* dentro de un filtro $F_n$, en [11] se demuestra experimentalmente que las CNN entrenadas con BSC pueden alcanzar, en muchos casos, resultados similares a las convoluciones convencionales. Sin embargo, a diferencia de las convoluciones estándar que

tienen $N \cdot M \cdot K^2$ parámetros libres, la variante BSC sólo tiene $N \cdot (M + K^2)$ parámetros ($N \cdot K^2$ parámetros para los patrones, y $M \cdot N$ parámetros para los escalares).

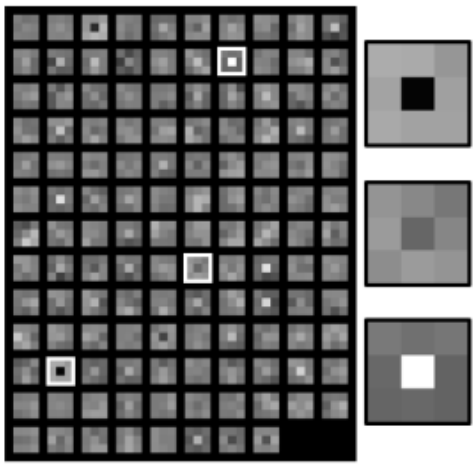


Figura 7: Visualización de un filtro de 128 *kernels* de dimensiones $3 \times 3$ de una red convolucional entrenada sobre un conjunto de imágenes [11].

También en [11] se demuestra que las convoluciones BSC pueden ser implementadas utilizando los siguientes tipos de convolución:

- Convolución puntual: La convolución puntual, también conocida como convolución *pointwise*, se trata de una convolución estándar caracterizada por contener *kernels* de tamaño $1 \times 1$.

- Convolución en profundidad: Esta convolución, también conocida como convolución *depthwise*, aplica un filtro a cada canal de entrada por separado. Cada canal sera tratado como una matriz de 2 dimensiones sobre la cual convolucionara un kernel, aislando así la información presente en cada canal.

En particular, para aplicar una convolución BSC se utiliza una convolución puntual, la cual genera un estado intermedio que servirá de entrada para una convolución en profundidad (Figura 8).

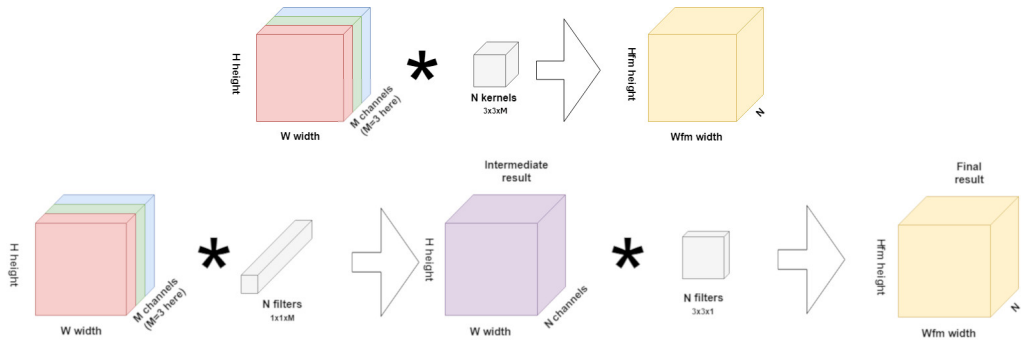



Figura 8: Comparación entre una convolución estándar(a) y un convolución separables por patrones(BSC)(b).

## 2.4. Funciones de costo

Dentro del área del aprendizaje profundo, la función de costos es una medida de desempeño para evaluar el rendimiento del modelo y determinar cuánto se debe ajustar el modelo para mejorar sus predicciones. Se calcula comparando las predicciones del modelo con los valores objetivo verdaderos, y luego se minimiza durante la etapa de entrenamiento del modelo. A continuación se detallan las dos funciones de costo utilizadas dentro de la tesina.

### Cross-Entropy

La entropía cruzada es la función de costo comúnmente utilizada en el aprendizaje automático; especialmente en problemas de clasificación y se define como:

$$CE = -\sum y - \log y` \tag{4}$$

donde $CE$ es la entropía cruzada, $y`$ es la predicción del modelo para una muestra dada e $y$ es el valor objetivo real de dicha muestra.

### Mutual-Channel Loss

La función de pérdida mutua por canal (MCLoss)[12] es una función de costos específicamente diseñada para resolver problemas de clasificación de

grano fino(Ver Sección 2.5). Esta función consta de dos componentes específicos, una componente de discriminación y una componente de diversidad que actúan sobre los *feature maps* de una capa intermedia de la arquitectura separando los distintos canales en grupos y especializandolos en la clasificación de una clase especifica (Figua 9).

Para explicar mejor el funcionamiento de esta función de costos es necesario dar algunas definiciones previas. Sea $f \in \mathbb{R}^{N*W*H}$ los *feature maps* extraídos de una capa intermedia de la red, donde H es la altura, W el ancho y N el número de mapas; y $\xi$ un hiperparametro que indica cuantos canales corresponden a cada clase, de forma tal que $\xi * c = N$ donde c es el número de clases dentro del conjunto de datos. Utilizamos $F_i \in \mathbb{R}^{\xi*W*H}$ para representar al grupo de canales correspondientes a la clase $i$ con $i \in 0, ..., c-1$

$$F_i = \{f_{i*\xi+1}, ..., f_{i*\xi+\xi}\} \tag{5}$$

Tomando $F = \{F_0, ..., F_{c-1}\}$ como los *feature maps* de la capa intermedia seleccionada, $Y$ como el valor objetivo real de la muestra e $Y'$ como la predicción de nuestro modelo se define a la función de costos MCLoss de la siguiente manera:

$$Loss(Y, F, Y') = L_{CE}(Y', Y) + \mu \cdot L_{MC}(F, Y) \tag{6}$$

$$L_{MC}(F, Y) = L_{dis}(F, Y) - \lambda \cdot L_{div}(F) \tag{7}$$

donde $L_{dis}$ y $L_{div}$ referencia a las componentes de discriminalidad y diversidad respectivamente y $\lambda$ y $\mu$ son dos constantes que se fijan durante la fase de entrenamiento.

La componente de discriminalidad $L_{dis}$ obliga a que los *feature maps* estén alineados con una clase especifica y que cada *feature map* correspondiente a una clase concreta contenga información útil. En otras palabras, $L_{dis}$ fomenta que los *feature maps* $F_i$ solamente contengan información relevante para la clasificación de la clase $i$ y penaliza aquellos *feature maps* de $F_i$ que no aporten información útil.

Por otra parte, la componente de diversidad $L_{div}$ es una medida de "distancia" aproximada para calcular la similitud entre los *feature maps* de una

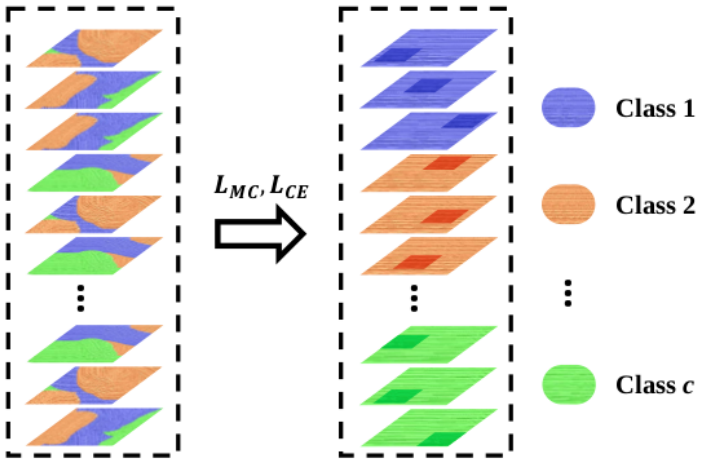



Figura 9: Comparación de los *feature maps* antes (izquierda) y después (derecha) de aplicar la función MCLoss, donde los *feature maps* se alinean con la clase, y cada uno atiende a diferentes partes discriminantes[12].

misma clase. El componente de diversidad hace que los *feature maps* de un grupo $F_i$ se diferencien entre sí. Es decir, los diferentes *feature maps* de una clase deben centrarse en diferentes regiones de la imagen en lugar de que todos los canales se centren en la región más discriminativa. De este modo, se reduce la información redundante mediante la diversificación de los *feature maps* de cada grupo y se ayuda a descubrir diferentes regiones discriminativas.

Definiciones más precisas acerca del funcionamiento de $L_{dis}$ y $L_{div}$ pueden encontrarse en [12]. Notar que la función MCLoss sólo se utiliza durante el entrenamiento y que sus componentes no están presentes en el momento de la inferencia.

## 2.5. Clasificación visual de grano fino y grueso

El análisis de imágenes de grano fino (FGIA, Fine-Grained Image Analysis) se centra en el tratamiento de objetos que son visualmente muy similares y que usualmente pertenecen a varias subcategorías de una misma categoría mas general (Figura 10). Mientras que en el análisis de imágenes genérico los objetos de destino pertenecen a categorías de grano grueso (Ej.: naranjas

y perros) que son visualmente muy diferentes entre sí; en FGIA los objetos presentan un alto grado de similitud y generalmente provienen de subcategorías de una categoría mas genérica (Ej.: diferenciación de especies de aves). Esta tarea suele ser mucho más difícil que la clasificación de grano grueso, ya que las diferencias visuales entre clases son a menudo sutiles y están localizadas en zonas específicas de los objetos. Los primeros trabajos se basaban en gran medida en anotaciones manuales de dichas zonas focalizándose principalmente en la mejor forma de localizar estas partes discriminativas de los objetos para luego llevar a cabo la clasificación [13].


Figura 10: Muestras de 5 categorías de imágenes diferentes, donde cada imagen pertenece a una subcategoria distinta [13].

Las principales dificultades de la clasificación de grano fino son:

- Variaciones entre las distintas clases pueden ser muy bajas debido al alto grado de similitud visual como se observa en la Figura 11.
- Existen grandes variaciones dentro de una misma clase debido a que las imágenes pueden ser tomadas en ángulos distintos y que algunas condiciones de iluminación pueden variar (como puede verse en la Figura 12).
- En la mayoría de los casos se requiere a expertos en la problemática a resolver para realizar el etiquetado lo que aumenta significativamente los costos y plazos de tiempo de este proceso.

Por estos motivos a lo largo de los años se han generado distintos modelos enfocados en esta problemática y conjuntos de datos específicos que muestran lo complejo que puede ser resolver este tipo de tareas [15, 16, 17, 18]. Sin embargo, se ha demostrado en [14] que los modelos convencionales logran buenos rendimientos en conjuntos de datos de clasificación de granularidad fina.


Figura 11: La diferencia entre distintas clases es limitada [14].

Figura 12: La variación dentro de una clase suele ser alta debido a la pose, la iluminación y el color. En cada fila se muestran imágenes de una misma clase [14].

# 3 Trabajos Relacionados

Los pequeños sistemas aéreos no tripulados (UAS, por sus siglas en inglés) han surgido como medios efectivos y asequibles para la recopilación de imágenes aéreas de cultivos a lo largo de ciclos completos de crecimiento. Estas plataformas de alto rendimiento se han convertido en herramientas indispensables en la agricultura de precisión y la investigación en fitomejoramiento al permitir la recopilación de datos de alta resolución en grandes campos de cultivo. Sin embargo, esta eficiencia mejorada en la captura de imágenes ha generado conjuntos de datos masivos, los cuales presentan desafíos en su análisis para obtener la valiosa información fenotípica requerida. Para abordar esta problemática, la mejora de cultivos ha impulsado la necesidad de desarrollar métodos sólidos de análisis de imágenes capaces de procesar grandes volúmenes de datos. Tradicionalmente, el análisis de imágenes y el aprendizaje automático han desempeñado un papel fundamental en la transformación de los datos de imágenes en información fenotípica específica, como la altura de las plantas [19, 20, 21] o el recuento de la población de plantas[22]. En general, las CNN profundas se aplican en una amplia variedad de tareas agrícolas y de fenotipado de plantas que no incorporan la genética subyacente que incluyan CNN profundas como por ejemplo, la detección de frutas[23, 24], identificación de malezas[25], clasificación de enfermedades de plantas [26], conteo de hojas [27], detección de estrés [28], entre otras tareas.

## 3.1. Conteo de panículas

En los países en desarrollo, los alimentos a base de cereales desempeñan un papel fundamental como alimentos básicos para adultos y alimentos de destete para lactantes. A nivel mundial, el sorgo se sitúa como el quinto grano más producido y juega un papel crucial en la seguridad alimentaria de estos países. Según la Organización de las Naciones Unidas para la Agricultura y la Alimentación (FAO), se estima que la producción mundial de cereales superará los tres mil millones de toneladas para el año 2050. Para lograr este objetivo, es esencial contar con un conocimiento profundo de las estadísticas de cultivos, como el área de cultivo y la producción de rendimiento para poder gestionar de manera efectiva los recursos y fomentar el desarrollo del sector agrícola, así como garantizar la seguridad alimentaria. El sorgo, una gramínea tropical, desempeña un papel clave en la provisión de nutrición

tanto para los seres humanos como para el ganado, especialmente en entornos con precipitaciones marginales. Un indicador crucial para evaluar el rendimiento del sorgo es el número de cabezas en diferentes disposiciones de ramificación. En el caso de cultivos como el sorgo, cada semilla plantada puede dar lugar a múltiples brotes, lo que significa que el número de cabezas por unidad de área puede ser mayor que el número de plantas. Esto plantea dificultades significativas en su estudio, especialmente en la reproducción a gran escala. Los datos fenotípicos, como los recuentos, el número de semillas por panícula y las medidas de tamaño (largo y ancho de la panícula), desempeñan un papel crucial en diversas evaluaciones, incluyendo la evaluación de la diversidad genética, la selección de nuevas variedades y la estimación de los rendimientos potenciales. En la Figura 13 podemos ver un ejemplo de la localización de panículas en un lote de sorgo para entender mejor la complejidad de esta tarea.

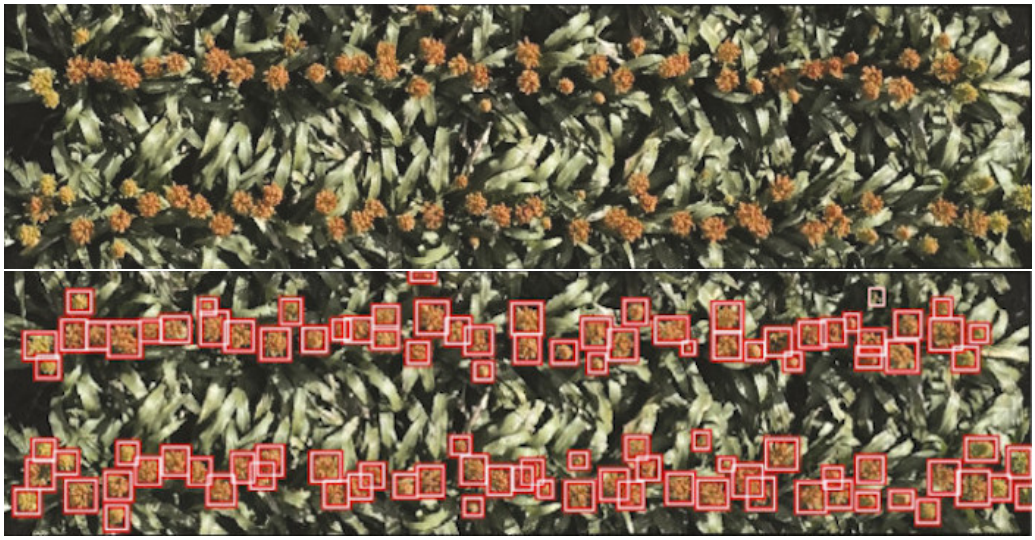


Figura 13: Localización de panículas de sorgo en imágenes aéreas.

La caracterización manual de las panículas ha demostrado ser una tarea tediosa y limitante en términos de eficiencia. Para abordar esta limitación, se han realizado diversos trabajos en busca de soluciones para automatizar y agilizar este proceso. En [29], se desarrolla un método de análisis basado en segmentación semántica para estimar el número de panículas de sorgo

en imágenes aéreas de alta resolución. En [30], se utilizan técnicas de TTA (Testing Time Augmentation) para mejorar la precisión de métodos de localización de panículas utilizando un modelo convolucional ResNet50[31]. Al momento de realizar una predicción estas técnicas realizan transformaciones de la imagen de entrada generando varias muestras y promediando las salidas del modelo. En [32], presentan una método de entrenamiento sobre una CNN basada en ResNet[31] que reduce la cantidad de datos etiquetados necesarios para entrenar un sistema de detección de panículas, permitiendo la generación de etiquetas sintéticas y reduciendo el costo de etiquetado manual.

## 3.2. Fenotipado de sorgo

Una de las tareas más básicas de fenotipado es determinar el cultivar, o la especie, de un grupo o de una parcela. A primera vista, la tarea de predecir el cultivar a partir de una imagen de una planta puede no parecer la pregunta biológicamente más convincente para responder: en el contexto del mejoramiento de plantas, el cultivar o las líneas parentales generalmente se conocen. Sin embargo, se puede usar un modelo de aprendizaje automático para determinar dónde pueden haberse producido errores en el proceso de plantación. Por ejemplo, la semilla puede estar mal etiquetada antes de la siembra, o las sembradoras pueden atascarse, depositando las semillas de manera no uniforme en un campo. Estos tipos de errores pueden causar problemas importantes al procesar datos de experimentos de campo a gran escala con cientos de cultivares y diseños de plantación de campo complejos. Además, es una tarea desafiante de reconocimiento visual, ya que una gran cantidad de cultivos altamente relacionados se cultivan simultáneamente, lo que genera un problema de clasificación con una baja variación entre clases.

En [33] se presenta un enfoque novedoso para entrenar una red neuronal convolucional profunda en la tarea de clasificación de cultivares, utilizando una arquitectura CNN de resolución múltiple. Esta arquitectura tiene dos ramas paralelas: una que es una rama de “escala global” de baja resolución y otra que es una rama de “escala local” de alta resolución. En la rama de escala global, la entrada es una versión redimensionada de la imagen original. En la rama local, hay cuatro modelos con pesos compartidos que ven recortes de la imagen original para extraer información utilizando una resolución de imagen mayor. Las salidas de las dos ramas luego se concatenan y se alimentan a través de una capa final completamente conectada. Cada

rama es implementada mediante una red convolucional ResNet50[31]. Una visualización de esta arquitectura puede verse en la Figura 14.

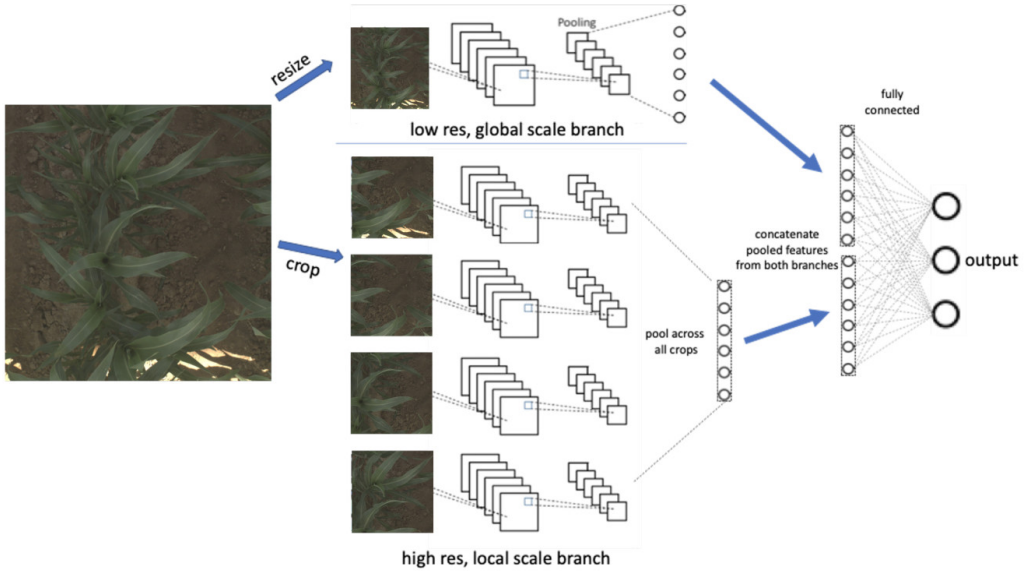



Figura 14: Arquitectura presentada en [33] para la clasificación de cultivares de sorgo.

Esta arquitectura es muy similar a la arquitectura piramidal de características de [34], que entrenó dos redes: una con imágenes del conjunto de datos ImageNet[35] para características de 'escala de objeto', y otra en imágenes del conjunto de datos Places[36], para características de 'escala de escena'.

Además, en [33] se presenta una nueva operación de pooling global conocida como *Outlier Pooling* la cual se basa en buscar outliers[1] dentro de los valores de entrada y calcular su promedio. De esta forma, se obtiene una operación que se adapta bien a configuraciones donde las características relevantes de un objeto de interés pueden estar intercaladas a lo largo de una imagen, como suele ocurrir en problemas de clasificación de grano fino.

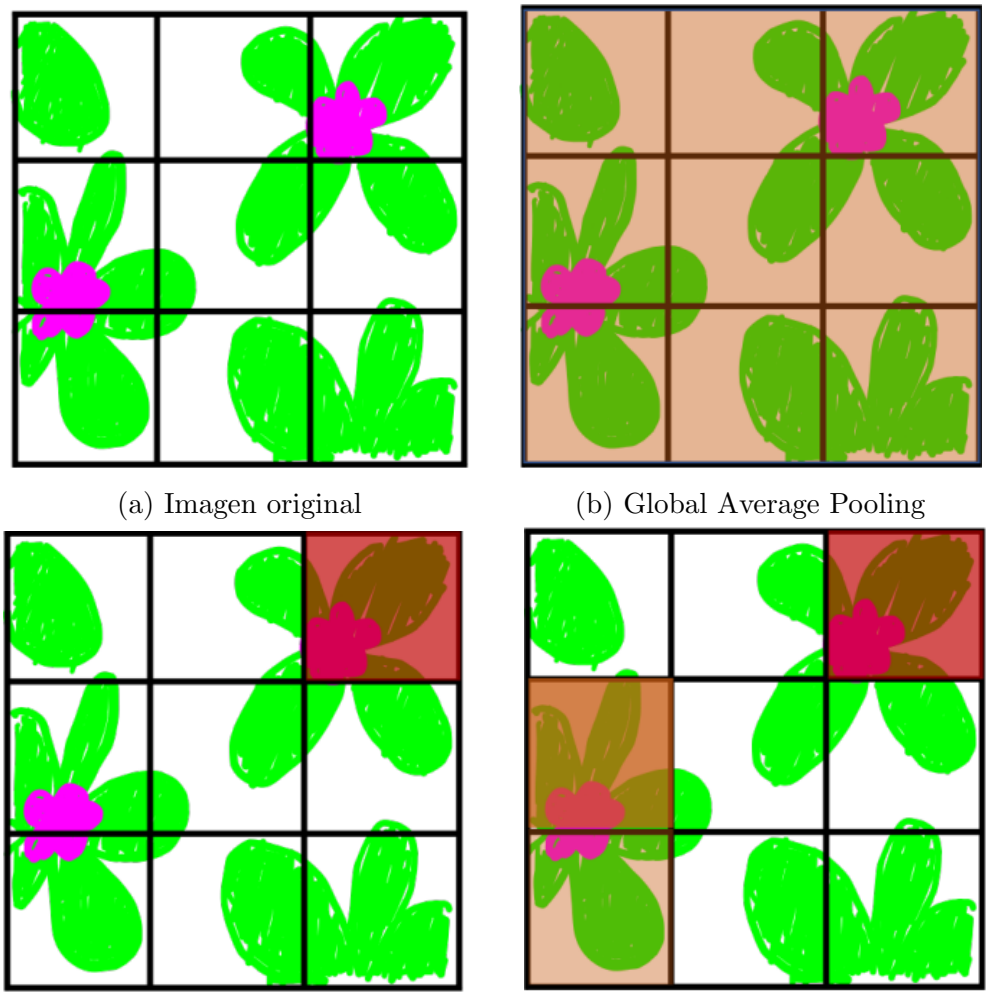
En la Figura 15, se ilustra este concepto de forma sencilla. Supongamos que tenemos un filtro que se activa en lugares con flores rosadas, como las que se muestran en la Figura 15 (a). A continuación, visualizamos la activación de este filtro cuando se pasa a través de diferentes operación de pooling.

[1]se consideran como outliers a aquellos valores que sean mayores a la media de la muestra mas dos veces la desviación estándar

En la Figura 15 (b), se representa la operación de pooling promedio global, donde las activaciones más altas se promedian con activaciones bajas que se corresponden a zonas donde no esta presente el objeto de interés. Por otro lado, en la Figura 15 (c), se muestra la operación de pooling máxima global, que solo considera la activación más alta, descartando las demás instancias. En contraste, en la Figura 15 (d), se aplica la operación de *Outlier Pooling*. Este enfoque tiene en cuenta únicamente las activaciones relevantes para el problema, evitando el sesgo generado por las operaciones anteriores. Es importante destacar que en la Figura 15, las ubicaciones sin activación no tienen color y la intensidad del color rojo se corresponde con la intensidad de la activación.

Profundizando un poco más la investigación en sorgo, en [37] se propone un proceso para comprender e identificar marcadores genéticos interesantes que controlan los rasgos visualmente observables en imágenes RGB. Este enfoque puede ser aprovechado por los programas de fitomejoramiento para comprender la relación entre los polimorfismos de un solo nucleótido (SNP, ubicaciones en el ADN del organismo que varían entre los diferentes miembros de la población), o grupos de SNP relacionados, y los fenotipos que impactan. El proceso se basa en el entrenamiento de una red convolucional ResNet50[31] para clasificar los cultivares, y luego analizar las partes específicas de la imagen en las que el modelo se enfoca al hacer una predicción. De esta manera, se pueden detectar automáticamente los rasgos visuales que caracterizan a una especie en particular. La Figura 16 ilustra cómo el modelo presentado en [37] resalta una región específica en la imagen que coincide con las características comúnmente utilizadas por los genetistas. Este enfoque ofrece una herramienta eficaz para identificar y comprender los marcadores genéticos asociados a los rasgos observables en las imágenes, lo que facilita el avance en la investigación agrícola y el desarrollo de nuevas variedades.

La mayoría de los modelos presentados en capitulo son intensivos en el uso de los recursos de computo lo que dificulta la posibilidad de nuevos dispositivos de IoT (Internet de las cosas) que emplean sistemas embebidos. En particular, los entornos agrícolas se ven afectados por condiciones limites como son la falta de conectividad y autonomía. En esta tesina se abordará este problema desarrollando un nuevo modelo basado en ResNet capaz de utilizarse en este ambiente del Agro.



(a) Imagen original

(b) Global Average Pooling

(c) Global Max Pooling

(d) Global Outlier Pooling

Figura 15: Visualización de distintas operaciones de pooling sobre un problema ficticio[33].

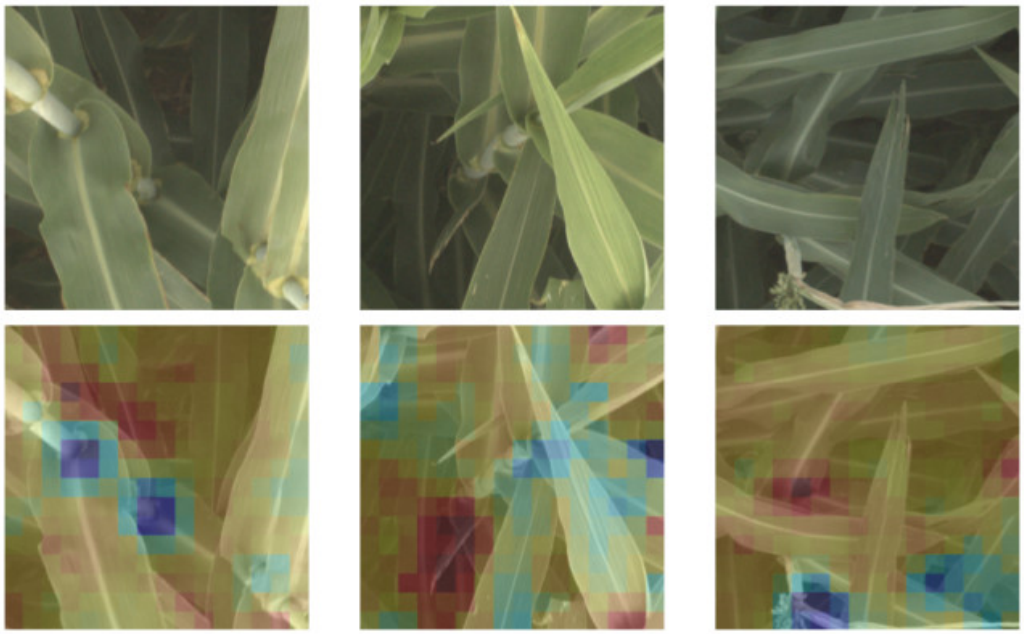


Figura 16: Mapa de calor sobre imágenes de sorgo presentado en [37] que resalta en azul las zonas claves de la imagen utilizadas por el modelo para realizar la predicción.

# 4 Materiales y métodos

En este capítulo, se detallan los conjuntos de datos y los modelos de redes convoluciones utilizados como así también nuestra red profunda convolucional denominada ResNet-BS.

## 4.1. Conjunto de datos

A continuación se detallan los conjuntos de datos utilizados en esta tesina.

### Cifar10

El conjunto de datos CIFAR-10[38] consta de 60000 imágenes a color de 32x32 en 10 clases, con 6000 imágenes por clase. Este conjunto de datos se divide en un conjunto de entrenamiento, que contiene exactamente 5000 imágenes seleccionadas al azar de cada clase, y un conjunto de prueba, conformado por las imágenes restantes (1000 imágenes de cada clase). Este conjunto de datos fue utilizado para pre-entrenar nuestros modelos.

### Sorghum-100

En 2016, la Agencia de Proyectos de Investigación Avanzada-Energía (ARPA-E) de Estados Unidos financió el desarrollo de la Plataforma de Referencia de Fenotipado de Agricultura Renovable(TERRA-REF)[39]. Este proyecto se basa en un sistema de monitoreo de última generación para recolectar información sobre el ciclo de crecimiento completo de más de un acre de cultivos con un conjunto de sensores de imagen de última generación, que incluye sensores estéreos RGB, térmicos, hiperespectrales, y escáner láser 3D. En el transcurso de sus primeros años de operación, la plataforma TERRA-REF[33] recopiló una gran cantidad de datos que capturaron el ciclo de crecimiento completo de un conjunto de 390 variedades de sorgo cuyos genomas han sido completamente secuenciados.

Dentro de este trabajo se utilizará principalmente el conjunto de datos Sorghum-100, el cual consiste en un subconjunto seleccionado de las imágenes RGB de sorgo capturadas durante los experimentos TERRA-REF, etiquetadas por variedad. Consta de 48.106 imágenes y 100 variedades de sorgo diferentes cultivados en junio de 2017 (las imágenes provienen de la mitad de la temporada de crecimiento) [33].

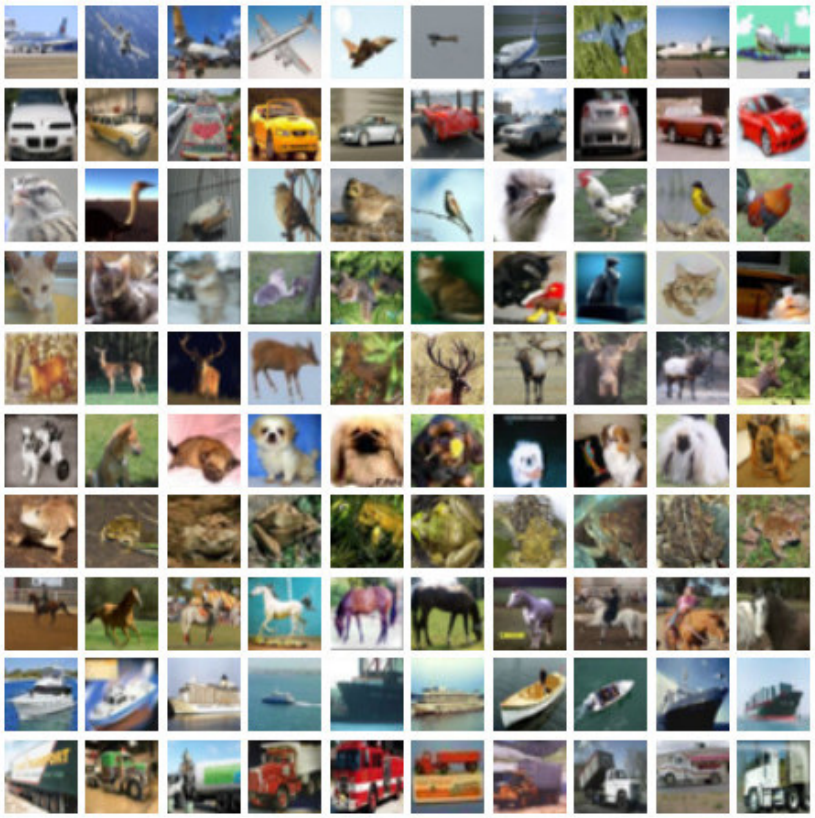


Figura 17: Ejemplo de imágenes de Cifar10, donde cada fila muestra imágenes de una misma clase.

El conjunto de datos Sorghum-100 se divide en un conjunto de datos de entrenamiento conformado por 22.635 imágenes, y un conjunto de datos de prueba con las imágenes restantes. Cada cultivar se cultivó en dos parcelas separadas dentro de un mismo lote para tener en cuenta las condiciones locales del suelo o del campo que podrían afectar el crecimiento de las plantas en una parcela en particular. Haciendo uso de esta división natural en los datos se crearon los conjuntos de entrenamiento y prueba: las imágenes para un cultivo determinado en el conjunto de datos de entrenamiento provienen de una parcela, mientras que las imágenes de prueba de ese mismo cultivar provienen de la otra parcela. Esto significa que un modelo no puede lograr un alto rendimiento al memorizar características que no son fenotipos significativos (por ejemplo, al memorizar patrones observados en la tierra)[33].

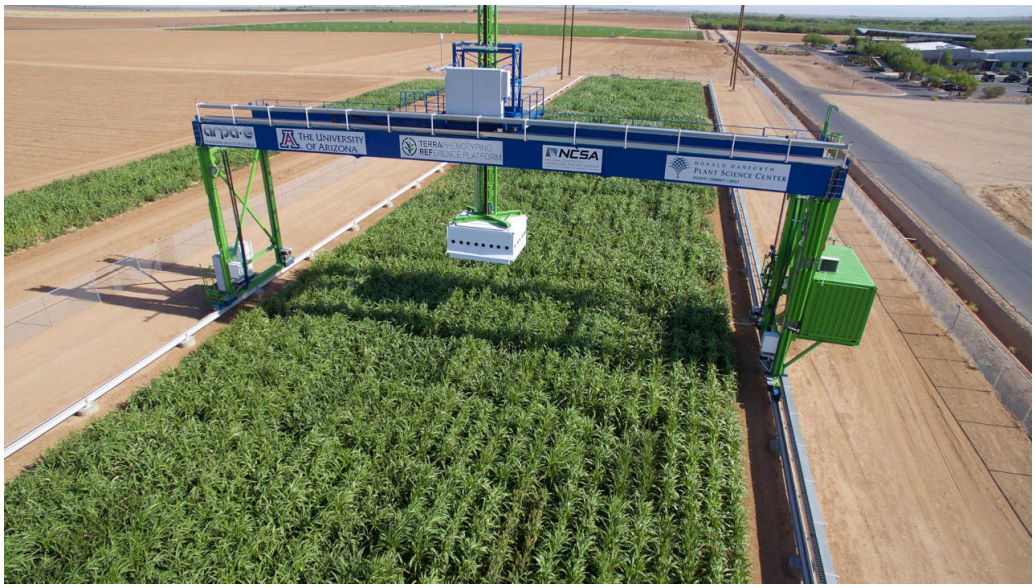



Figura 18: El escáner de campo de TERRA-REF en Maricopa, Arizona, con sorgo cultivado en el campo.

## 4.2. Arquitecturas

### ResNet

ResNet [31] es una de las arquitecturas de clasificación de imágenes más utilizada. Supuso una mejora notable en el momento en que se introdujo y sigue sirviendo como arquitectura de referencia para algunos análisis, o como línea base en trabajos que introducen nuevas arquitecturas [40].

Diversos estudios muestran que la profundidad de la red tiene una importancia crucial y los principales resultados en el desafiante conjunto de datos ImageNet[41] explotan modelos "muy profundos" con una profundidad de hasta 30 capas. Impulsada por la importancia de la profundidad, surge una pregunta: ¿Agregar más capas es suficiente para obtener mejores resultados?

Cuando se entrenan redes más profundas, se ha puesto de manifiesto un problema de degradación: al aumentar la profundidad de la red, el rendimiento se satura (lo que podría no sorprender) y luego se degrada rápidamente. Inesperadamente, esta degradación no está causada por el sobreajuste y añadir más capas a un modelo puede conducir a un mayor error de entrenamiento como se indica en [31].

En [31] se aborda el problema de la degradación introduciendo un marco

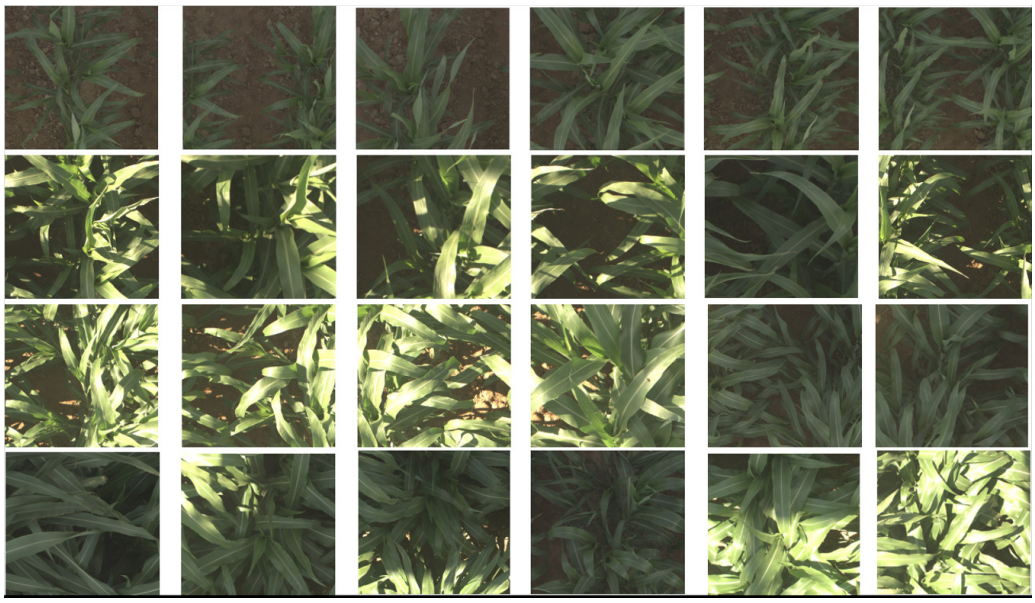


Figura 19: Ejemplo de imágenes del conjunto Sorghum-100. Cada fila muestra un mismo fenotipo de sorgo.

de aprendizaje residual profundo. En lugar de esperar que cada una de las capas apiladas se ajuste directamente a un mapeo subyacente deseado, ResNets deja explícitamente que estas capas se ajusten a un mapeo residual. Formalmente, denotando el mapeo subyacente deseado como $H(x)$, las capas no lineales apiladas ajustan otro mapeo de $F(x) := H(x) - x$. El mapeo original se redefine como $F(x) + x$.

La formulación de $F(x) + x$ puede realizarse mediante redes neuronales con "conexiones de acceso directo" (Figura 20) que son aquellas que se saltan una o más capas.

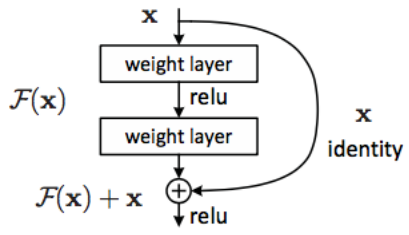



Figura 20: Residual block.

En el caso de ResNet, las conexiones de acceso directo simplemente rea-

lizan un mapeo de identidad y sus salidas se añaden a las salidas de las capas apiladas. En esta arquitectura se utiliza el aprendizaje residual luego de apilar un pequeño grupo de capas. En la Figura 20 se muestra un bloque de construcción que puede ser formalmente definido como:

$$y = F(x, \{W_i\}) + x \tag{8}$$

donde $x$ e $y$ son los vectores de entrada y salida de las capas consideradas respectivamente y la función $F(x, \{W_i\})$ representa el mapeo residual que debe aprenderse. Notar que las dimensiones de $x$ y $F$ deben ser iguales en la Ecuación 8. Si no es el caso (por ejemplo, al cambiar los canales de entrada/salida), podemos realizar una proyección lineal $W_s$ por las conexiones de acceso directo para igualar las dimensiones. Esta proyección se implementa como una convolución de un kernel de tamaño $1 \times 1$ y paso de aplicación de $2 \times 2$:

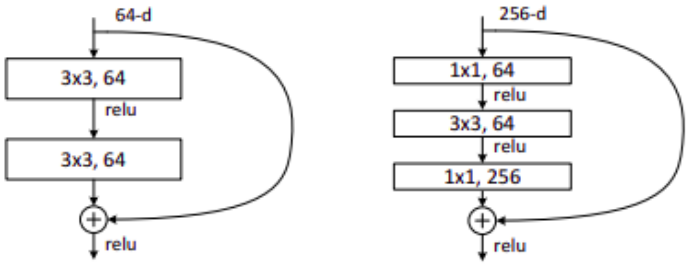
$$y = F(x, \{W_i\}) + W_s x \tag{9}$$

En la Figura 21 se muestra la arquitecturas de ResNet de distintas profundidades [31]. Los bloques de construcción se muestran entre paréntesis, con el numero de bloques apilados. Podemos ver mas en detalle estos bloques en la Figura 22 y como se aplica la conexión residual.

| layer name | 18-layer | 50-layer |
|---|---|---|
| conv1 | 7×7, 64, stride 2 | |
| | 3×3 max pool, stride 2 | |
| conv2_x | [3×3, 64; 3×3, 64] ×2 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | [3×3, 128; 3×3, 128] ×2 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 |
| conv4_x | [3×3, 256; 3×3, 256] ×2 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 |
| conv5_x | [3×3, 512; 3×3, 512] ×2 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | average pool, 1000-d fc, softmax | |

Figura 21: Arquitectura de los modelos ResNet18 y ResNet50.

Usualmente, para hacer referencia a un modelo particular se utiliza el nombre en función de la profundidad (Ej. ResNet18). En la Figura 23a se puede ver de manera gráfica la arquitectura del modelo ResNet50 utilizado en esta tesina.




(a) ResNet18 (b) ResNet50

Figura 22: Bloques de construcción de ResNet.

ResNet se ha convertido en una de las arquitecturas de clasificación de imágenes más populares que ha logrado resultados sobresalientes en una variedad de problemas de clasificación de imágenes, incluyendo una precisión de 0.8 en el desafío ImageNet [40]. Al momento de escribir esta tesina, el trabajo que introdujo esta arquitectura cuenta con más de 150.000 citas y fue la arquitectura utilizada como base en la investigación que presenta al conjunto de datos Sorghum-100 [33].

## ResNet BS

Para este trabajo se seleccionó a ResNet como la arquitectura base a modificar. En particular, nos enfocamos en las arquitecturas ResNet18 y ResNet50, ambas conformadas por una convolución inicial cuyos *kenels* tiene un tamaño de $7 \times 7$ y convoluciones con *kernels* de $3 \times 3$ y $1 \times 1$ distribuidas a lo largo de su estructura (Figura 21). Para optimizar su desempeño, se reemplazaron las convoluciones de $3 \times 3$ y $7 \times 7$ con convoluciones BSC. Las convoluciones BSC han demostrado ser efectivas para reducir la cantidad de parámetros y la cantidad de operaciones de punto flotante (FLOPs) en una arquitectura. De aquí en mas llamaremos a esta variante ResNet-BS (Figura 23b).

La Tabla 1 muestra que la disminución obtenida fue del 83 % en la cantidad de parámetros y del 77 % en GFLOPs para ResNet18, mientras que para ResNet50 se logró una reducción del 39 % en la cantidad de parámetros y del 37 % en FLOPs. Se puede observar que las modificaciones propuestas tienen un menor impacto sobre la arquitectura ResNet50. Esto se debe a que este modelo, a diferencia de la arquitectura ResNet18, contiene convoluciones con *kernels* de tamaño $1 \times 1$. Por la forma en que las convoluciones

BSC simplifican una convolución estándar y por el hecho de que las convoluciones puntuales son un bloque integral de BSC, no es posible utilizarlas para simplificar una convolución con tamaño de *kernel* de $1 \times 1$. Más aun, partiendo de que una convolución estándar cuenta con $N \cdot M \cdot K^2$ parámetros y la variante BSC con $N \cdot (M + K^2)$, se puede demostrar que solamente se obtiene una reducción en el numero de parámetros si el tamaño del *kernel* es mayor o igual a 2.

Las versiones modificadas fueron creadas para mejorar la eficiencia de los modelos al reducir la cantidad de parámetros, FLOPs y latencia. Esto proporciona flexibilidad para seleccionar el modelo más adecuado según los requisitos del problema en cuestión. Sin embargo, el uso de convoluciones BSC suele tener un impacto negativo en la capacidad del modelo para ajustarse a los datos, tal como se ha observado en los resultados experimentales publicados en [11] y presentados en el repositorio de la investigación[2]. En esta tesina se buscará utilizar estas convoluciones de forma que se minimice el impacto en el desempeño del modelo.

| Arquitectura | Parametros | FLOPs |
|---|---:|---:|
| ResNet18 | 11.6M | 9.5G |
| ResNet18-BS | 1.9M | 2.1G |
| ResNet50 | 25.5M | 21.5G |
| ResNet50-BS | 15.5M | 13.5G |

Tabla 1: Arquitecturas, parametros y FLOPs de ResNet18, ResNet50, ResNet18-BS y ResNet50-BS

Para tratar de mitigar la pérdida de rendimiento ocasionada por las modificaciones realizadas se utiliza la función de costos MCLoss. El uso de esta función de costos no agrega ningún tipo de retardo o computo sobre el modelo al momento de llevarlo a producción, ya que solamente esta presente durante el proceso de entrenamiento. En este trabajo, utilizamos el enfoque presentado en [12] para el modelo ResNet. Seleccionamos la última capa convolucional como capa intermedia para aplicar esta función de costos. En lugar de asignar los canales de manera uniforme, decidimos distribuir la cantidad de canales en función de la cantidad de *feature maps* de salida de dicha capa. La asignación de la cantidad de canales se realizo en base al protocolo experimental presentado en [12]. En ResNet18, se repartieron los 512 *feature maps* de la capa intermedia asignando 5 *feature maps* a cada una

[2]https://github.com/zeiss-microscopy/BSConv

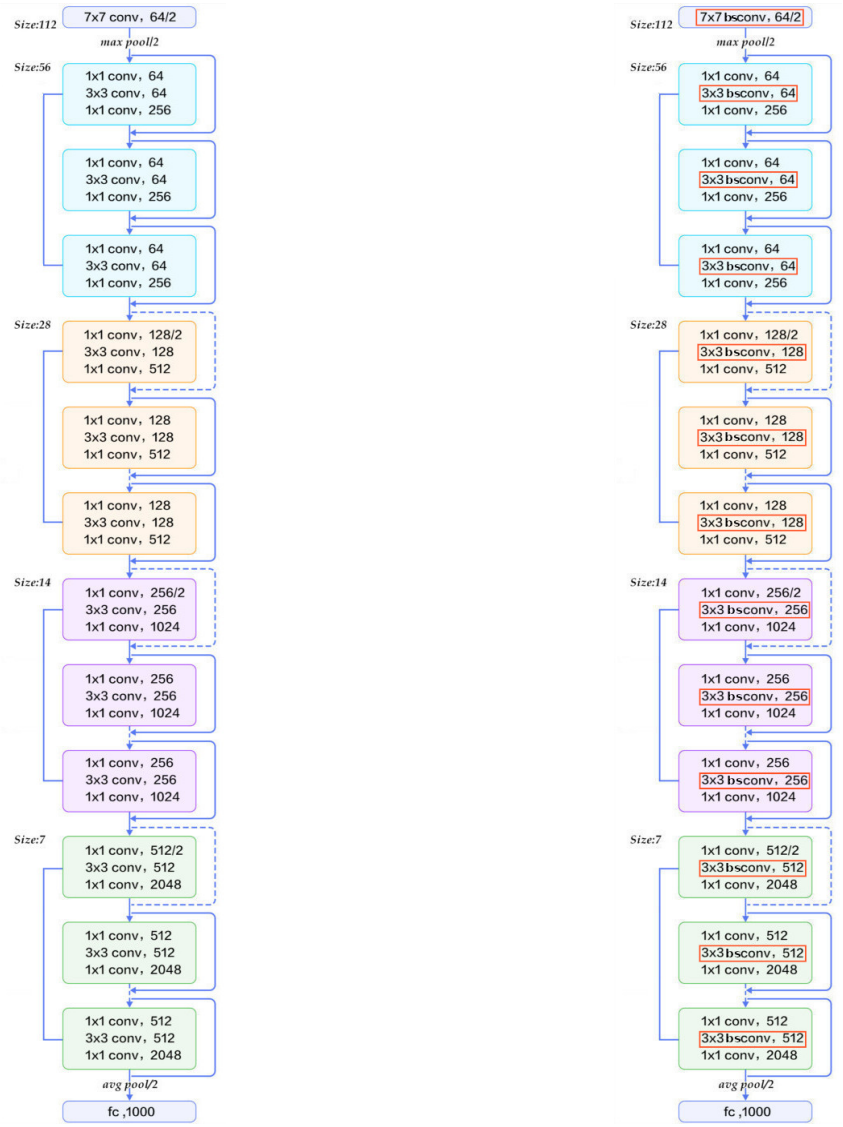
de las primeras 88 clases y 6 para las 12 clases restantes de nuestro problema. En la caso de ResNet50, la capa intermedia cuenta con 2048 *feature maps*, por lo que se asignaron 20 para las primeras 52 clases y 21 para las clases restantes.

Una de las principales ventajas de las modificaciones propuestas es la reducción en la latencia del modelo al momento de realizar inferencias. Esta reducción puede tener un impacto significativo en el presupuesto de un proyecto, ya que una misma computadora podría procesar más imágenes en el mismo periodo de tiempo o incluso se posibilita la reducción de costos al utilizar un hardware con menos prestaciones.

## Sistemas embebidos

En aplicaciones de inteligencia artificial, no siempre es viable utilizar computadoras de gran tamaño y alto consumo de energía. Para estos casos, se utilizan computadoras de placa simple, también conocidas como monoplacas. Estas computadoras están construidas en una única placa de circuito y cuentan con microprocesadores, memoria, puertos de entrada/salida y otras características funcionales. El desarrollo tecnológico de las monoplacas y su acercamiento a las prestaciones de los ordenadores personales ha potenciado su uso en el campo de la inteligencia artificial. Además, algunos modelos de monoplacas tienen tanto CPU como GPU, lo que aumenta su aplicabilidad en el aprendizaje profundo [42].

Dentro de este ecosistema de monoplacas, las placas NVIDIA Jetson son una excelente opción para aplicaciones como la clasificación de imágenes, la detección de objetos, la segmentación y el procesamiento del habla. En esta tesina, utilizamos dos modelos de este ecosistema, Jetson Nano (Ver Fig. 24a) y Jetson Xavier NX (Ver Fig. 24b), para medir el impacto de nuestras modificaciones. La placa Jetson Nano tiene un procesador ARM Cortex-A57 de cuatro núcleos a 1.43 GHz, una placa de video Maxwell de 128 núcleos a 921 MHz y un peso de 141 gramos[43]. Por otro lado, la placa Jetson Xavier NX tiene un procesador NVIDIA Carmel ARMv8.2 de seis núcleos a 1.9 GHz, una placa de video Volta con 384 núcleos a 1100 MHz y un peso de



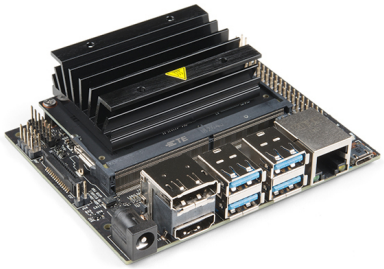

(a) Arquitectura ResNet50.

(b) Arquitectura ResNetBS-50.

Figura 23: Visualización de las arquitecturas ResNet50 y ResNet50-BS. Los recuadros rojos representan las convoluciones BSC.

172 gramos[44]. En ambos casos, utilizamos el software del sistema JetPack 4.6[3].

[3] `https://developer.nvidia.com/embedded/jetpack`



(a) Placa Jetson Nano con su entorno de desarrollo.

(b) Placa Jetson Xavier NX con su entorno de desarrollo.

# 5 Experimentación y resultados

En este capítulo se encuentra la parte experimental de la tesina. Particularmente, nos enfocaremos en la arquitectura ResNet en combinación con la convolución BSC para tratar de reducir la cantidad de recursos utilizados por el modelo evitando una degradación en las métricas del problema. Para finalizar brindamos una comparación del rendimiento sobre sistemas embebidos típicamente utilizados para la puesta en producción de sistemas de aprendizaje profundo.

## 5.1. Análisis del conjunto de datos Sorghum-100

Cada imagen dentro de este conjunto posee una resolución de $1024 \times 1024$ y le corresponde 1 (y solo 1) de las 100 posibles variedades de sorgo. Realizar experimentos con imágenes que tienen una resolución tan alta requiere un enorme poder de computo. Por este motivo, se optó por escalar las imágenes a la resolución $512 \times 512$ para poder reducir esta demanda de computo como en [33].

Se puede observar en la Figura 19 que el conjunto de datos presenta imágenes con problemas de iluminación, como sombras o zonas saturadas, lo cual complica el análisis. Por lo tanto, se aplicó un pre-procesamiento utilizando el método CLAHE [45] (Contrast Limited Adaptive Histogram Equalization) para mejorar las condiciones de iluminación.

Brevemente, el método CLAHE se basa en la técnica de ecualización de histograma adaptativa limitada por contraste y usualmente es aplicado sobre imágenes en blanco y negro. Este método divide la imagen en bloques más pequeños y calcula un histograma local para cada bloque, lo que permite obtener la distribución de los niveles de gris en esa región específica. Luego, aplica una función de distribución acumulativa para ajustar los valores de los píxeles según la distribución local de cada bloque.

La clave del CLAHE radica en su capacidad para limitar el contraste en cada bloque, evitando así el exceso de realce y la amplificación del ruido. Esto se logra mediante la introducción de un parámetro de "limitación de contraste"que evita que los valores de los píxeles se extiendan más allá de ciertos límites predefinidos. Al limitar el contraste, se asegura la preservación de los detalles y texturas de la imagen, evitando la pérdida de información en áreas brillantes o sombreadas. Esta perdida de información es muy común

en otros métodos menos sofisticados. Un ejemplo de esto puede observarse en la Figura 25.

(a) Original (b) Ecualización simple (c) CLAHE

Figura 25: Ejemplo de aplicación del método CLAHE.

Es importante destacar que el método CLAHE se aplica generalmente en un solo canal de la imagen, lo que significa que se utiliza principalmente en imágenes en blanco y negro. Sin embargo, es posible adaptarlo para su uso en imágenes en color mediante la conversión de la imagen original a un espacio de color adecuado y aplicando el método CLAHE solo en el canal necesario. En este trabajo, convertimos las imágenes al espacio de color HSV para luego aplicar este método sobre el canal V. En la Figura 26 se pueden observar algunos ejemplos de los resultados obtenidos.

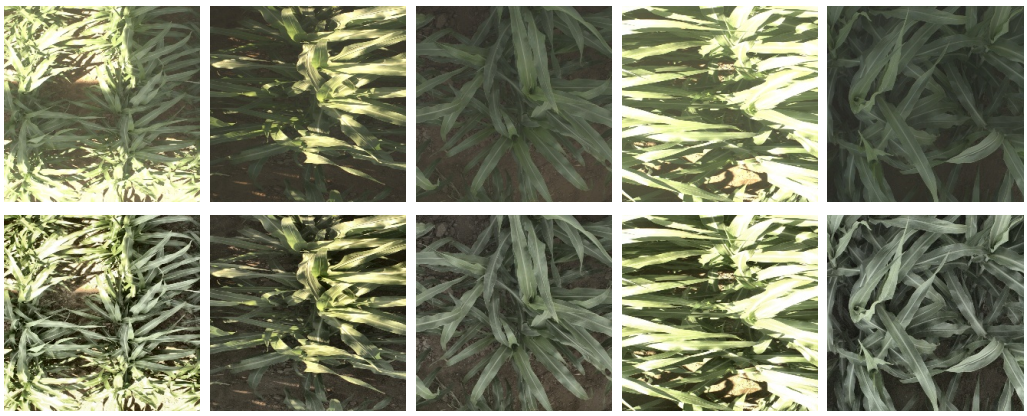


Figura 26: Ejemplos de los resultados obtenidos mediante el método de preprocesamiento sobre imágenes de Sorghum-100. La primera fila muestra las imágenes originales y la segunda fila los resultados obtenidos.

Si analizamos la distribución de las clases se puede observar un desbalance entre ellas. En particular, la clase con menos elementos cuenta con 134 ejemplos, mientras que la clase mas numerosa cuenta con 298 imágenes. De la Figura 27 se desprende que la media es de 221 imágenes. Para contrarrestar esto, utilizamos un método similar al implementado en [29]. Durante el entrenamiento se lleva a cabo un proceso de balanceo de datos que consiste en asignar a cada muestra un valor en función de la clase a la que pertenece. Este valor representa la probabilidad de que una imagen sea utilizada durante una época de entrenamiento y puede ser expresado como $0{,}01/k$ donde $k$ representa la cantidad de ejemplos de la clase a la que pertenece. De esta forma, imágenes pertenecientes a clases con menos ejemplos aparecerán con mayor frecuencia durante el proceso de entrenamiento que imágenes pertenecientes a las clases con más ejemplos. Como resultado, en cada época se entrena con un conjunto de datos balanceado. Además, las muestras que no son seleccionadas en una época en particular pueden ser utilizadas en épocas posteriores.

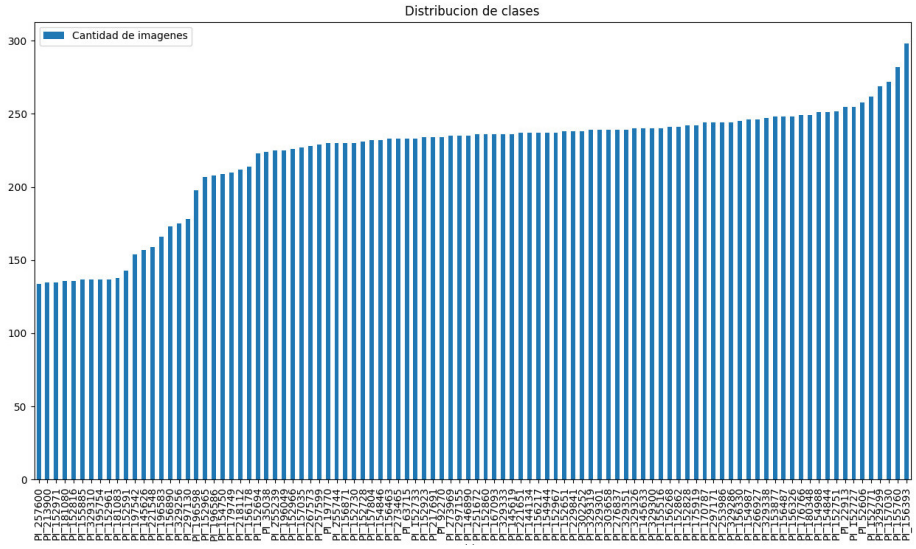



Figura 27: Histograma de distribución de las clases Sorghum-100.

## 5.2. Protocolo experimental

Los experimentos se llevaron a cabo en un servidor que cuenta con un procesador AMD EPYC 7281 y una GPU GeForce RTX 3090. Como framework principal de desarrollo se utilizó Pytorch[4] v1.11. Durante el entrenamiento se utilizó descenso estocástico por el gradiente[5] con un momentum de 0.9, una tasa de aprendizaje de 0.05 y un decaimiento de los pesos de $2 * 10^{-5}$. La tasa de aprendizaje inicial se fija en 0.05 y se reduce en un factor de 0.1 en las épocas 40 y 60. Los entrenamientos tuvieron una duración de 75 épocas. Normalizamos por medias de canal y desviaciones estándar utilizando estadísticas de ImageNet y se utilizaron técnicas para aumentar las imágenes tales como espejado horizontal y/o vertical de manera aleatoria con un 50 % de probabilidad junto con ajustes de brillo o contraste aleatorios con un 30 % de probabilidad. Para realizar estas transformaciones se utilizo la librería albumentations [46], utilizada también en [30].

En este trabajo nos enfocamos en el conjunto de datos Sorghum-100 (Ver Sec. 4.1). Este conjunto de datos consta de una división natural de entrenamiento y test debido a que se recolectaron imágenes de 2 lotes distintos. Esto significa que un modelo no puede lograr un alto rendimiento al memorizar características que no son fenotipos significativos. Adicionalmente, se apartaron un 10 % de los datos de entrenamiento como conjunto de validación con el propósito de seleccionar la versión óptima de los pesos del modelo, evaluándolo al final de cada época y reducir así el riesgo de sobreajuste.

Para esta tesina se pre-entrenaron las redes sobre el conjunto de datos cifar10 de forma similar a lo realizado en [11]. Dicho pre-entrenamiento se llevó a cabo durante 200 épocas utilizando descenso estocástico por el gradiente con un momentum de 0,9 y un decaimiento de pesos de $10^{-4}$. La tasa de aprendizaje inicial se fija en 0.1 y se reduce en un factor de 0.1 en las épocas 100, 150 y 180. Las imágenes se aumentan con espejados horizontales aleatorios y desplazamientos aleatorios de hasta 4 píxeles para evitar que los modelos se sobreajusten.

Para evaluar modelos de aprendizaje profundo sobre el conjunto de datos Sorghum-100 se realizaron 5 entrenamientos y se reportan el promedio y la desviación estándar de los resultados obtenidos sobre el conjunto de test.

[4] `https://pytorch.org/`

[5] `https://pytorch.org/docs/stable/generated/torch.optim.SGD.html`

## Métricas

A continuación se describen las métricas utilizadas para medir el desempeño de nuestro modelo sobre el problema de clasificación de fenotipos de sorgo:

- Accuracy: Es el porcentaje de predicciones correctas realizadas por un modelo. Se calcula dividiendo el número de predicciones correctas entre el número total de predicciones.

$$Accuracy = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}} \tag{10}$$

- Precisión: Para un problema de clasificación binaria, la precisión es la proporción de predicciones positivas correctas en relación con todas las predicciones positivas realizadas por el modelo. En un problema de clasificación de múltiples clases, la precisión se obtiene al calcular la precisión individual para cada clase y luego calcular el promedio de estas precisiones.

$$Precision = \frac{1}{n}\sum_{i=1}^{n} \frac{\text{Verdaderos positivos}_i}{\text{Verdaderos positivos}_i + \text{Falsos positivos}_i} \tag{11}$$

- Recall: Para un problema de clasificación binaria, el recall (también conocido como sensibilidad o exhaustividad) se refiere a la proporción de verdaderos positivos entre la suma de verdaderos positivos y falsos negativos. Representa la capacidad del modelo para identificar correctamente los ejemplos positivos. En un problema de clasificación de múltiples clases, el recall se obtiene al calcular el promedio de los recalls individuales de cada clase.

$$Recall = \frac{1}{n}\sum_{i=1}^{n} \frac{\text{Verdaderos positivos}_i}{\text{Verdaderos positivos}_i + \text{Falsos negativos}_i} \tag{12}$$

- Top 5: Es una métrica que indica la capacidad de un modelo de aprendizaje computacional para clasificar correctamente las cinco principales opciones o categorías más relevantes. Se utiliza comúnmente en tareas de clasificación donde se deben seleccionar múltiples opciones o se busca identificar las mejores opciones dentro de un conjunto más amplio. El Top 5 se calcula evaluando si la clase correcta está dentro de las

cinco clases con mayor confianza que predijo nuestro modelo. El valor representa a la cantidad de predicciones correctas dentro de las cinco clases principales sobre el numero total de predicciones.

- Latencia: es el tiempo que transcurre desde que se proporciona una entrada al modelo hasta que se obtiene la salida correspondiente. En otras palabras, es el tiempo de procesamiento que le toma a un modelo realizar una inferencia.

## 5.3. Linea Base

Para establecer una linea base se entrenaron los modelos ResNet18 y ResNet50 sobre los datos del conjunto Sorghum-100 utilizando la función de costos de entropía cruzada.

| Arquitectura | Accuracy | Top 5 | Recall | Precisión |
|---|---|---|---|---|
| ResNet50 [33] | 0.721 | - | - | - |
| ResNet18 | $0{,}713 \pm 0{,}010$ | $0{,}923 \pm 0{,}003$ | $0{,}717 \pm 0{,}011$ | $0{,}719 \pm 0{,}011$ |
| ResNet50 | $0{,}734 \pm 0{,}013$ | $0{,}934 \pm 0{,}002$ | $0{,}733 \pm 0{,}013$ | $0{,}738 \pm 0{,}012$ |

Tabla 2: Resultados de arquitecturas ResNet entrenados sobre el conjunto de datos Sorghum-100.

Como podemos observar en la Tabla 2, se obtienen resultados similares a los introducidos en [33]. Esto es de esperarse, ya que se trabaja con la misma resolución de imagen y con modelos similares.

## 5.4. ResNet-BS

Como se describió en la sección Sec. 4.2, existen diversas ventajas para introducir a las convoluciones BSC. Sin embargo, en [11] se muestra que estas modificaciones usualmente pueden acarrear una degradación en el rendimiento del modelo. Por lo tanto, se decidió entrenar las arquitecturas ResNet18-BS y ResNet50-BS para examinar su desempeño sobre nuestro problema de datos y analizar si existe o no un decremento en los resultados obtenidos.

Como se puede observar en la Tabla 3, al aplicar las convoluciones BSC sobre la arquitectura ResNet50 se obtiene una degradación en las métricas del modelo. En esencia las convoluciones BSC simplifican a una convolución

| Arquitectura | Accuracy | Recall | Precisión |
|---|---|---|---|
| ResNet18 | $0{,}713 \pm 0{,}010$ | $0{,}717 \pm 0{,}011$ | $0{,}719 \pm 0{,}011$ |
| ResNet18-BS | $0{,}694 \pm 0{,}017$ | $0{,}698 \pm 0{,}019$ | $0{,}700 \pm 0{,}016$ |
| ResNet50 | $0{,}734 \pm 0{,}013$ | $0{,}733 \pm 0{,}013$ | $0{,}738 \pm 0{,}012$ |
| ResNet50-BS | $0{,}690 \pm 0{,}017$ | $0{,}693 \pm 0{,}018$ | $0{,}696 \pm 0{,}017$ |

Tabla 3: Comparacion de la precisión de la arquitectura ResNet y ResNet-BS sobre el problema de clasificación de imágenes Sorghum-100.

estándar mediante el agregado de una restricción que implica que los distintos kernels de un filtro puedan ser obtenidos mediante la multiplicación de un único kernel y un escalar lo cual esta entorpeciendo el entrenamiento del modelo.

A continuación, mitigamos esta degradación mediante la utilización de la función de costos MCLoss. De esta forma, podríamos conservar el rendimiento del modelo original y aprovechar las reducciones en el computo derivadas de BSC.

## Análisis de la función de costos

Se ha cuestionado el uso de la función de costos de entropía cruzada para problemas de clasificación de grano fino en diversos trabajos [12, 47, 48]. Esto motivó a la utilización de una función de costos distinta para llevar a cabo el entrenamiento. En nuestro trabajo, hemos experimentado con la función de costos MCLoss para comparar su rendimiento con la función de costos de entropía cruzada.

En cuanto a la aplicación de MCLoss, esta función se aplica sobre los *feature maps* que sirven como entrada a la operación de *average pooling* que se encuentran en las últimas capas de la arquitectura ResNet. La asignación de canales se realiza de manera no uniforme, similar a lo planteado en los experimentos de [12]. Para la arquitectura ResNet18, se asignan 5 canales a 88 clases y 6 a las 12 clases restantes. Para la arquitectura ResNet50, se asignan 20 canales a 52 clases y 21 canales a las 48 restantes. La cantidad de canales asignados se ha decidido teniendo en cuenta la cantidad de clases de nuestro problema (100) y la cantidad de *feature maps* de cada arquitectura (512 y 2048 para las arquitecturas ResNet18 y ResNet50 respectivamente).

Por último, se han utilizado los valores de hiperparámetros $\mu = 5 \times 10^{-3}$ y $\lambda = 1{,}5$ para ResNet18 y $\mu = 5 \times 10^{-3}$ y $\lambda = 10$ para ResNet50. Así, hemos comparado el rendimiento de MCLoss con la función de costos de entropía

cruzada en el entrenamiento de la arquitectura ResNet en nuestro problema de clasificación de grano fino.

| Arquitectura | Función | Accuracy | Recall | Precision |
|---|---|---|---|---|
| ResNet18 | CELoss | $0{,}713 \pm 0{,}010$ | $0{,}717 \pm 0{,}011$ | $0{,}719 \pm 0{,}011$ |
| ResNet18 | MCLoss | $0{,}732 \pm 0{,}025$ | $0{,}736 \pm 0{,}024$ | $0{,}740 \pm 0{,}024$ |
| ResNet50 | CELoss | $0{,}734 \pm 0{,}013$ | $0{,}733 \pm 0{,}013$ | $0{,}738 \pm 0{,}012$ |
| ResNet50 | MCLoss | $0{,}752 \pm 0{,}019$ | $0{,}752 \pm 0{,}020$ | $0{,}752 \pm 0{,}025$ |

Tabla 4: Comparación de las funciones de costos MCLoss y entropia cruzada(CELoss) sobre el problema Sorghum-100.

La función MCLoss ha demostrado brindar mejoras en comparación con la función de costos de entropía cruzad, tal como se puede observar en la Tabla 4. En particular, los modelos ResNet50 y ResNet18 lograron una precisión de 0.752 y 0.732 respectivamente al aplicar esta función. Cabe destacar que los resultados obtenidos por ResNet18 utilizando la función MCLoss fueron ligeramente superiores a los resultados publicados previamente para el modelo ResNet50 en [33].

En [12] se realizan experimentos similares con la arquitectura ResNet18 que mostraron mejoras significativas, lo que nos impulsa a pensar en que sería posible utilizar esta función de costos para contrarrestar la degradación en el rendimiento obtenida en la Sección 5.4.

Tratando de mejorar los resultados presentados en la Sección 5.4 se entrena la arquitectura ResNet-BS utilizando la función de costos MCLoss.

| Arquitectura | Función | Accuracy | Recall | Precision |
|---|---|---|---|---|
| ResNet18-BS | MCLoss | $0{,}708 \pm 0{,}012$ | $0{,}712 \pm 0{,}009$ | $0{,}713 \pm 0{,}012$ |
| ResNet18 | MCLoss | $0{,}732 \pm 0{,}025$ | $0{,}736 \pm 0{,}024$ | $0{,}740 \pm 0{,}024$ |
| ResNet50-BS | MCLoss | $0{,}737 \pm 0{,}014$ | $0{,}732 \pm 0{,}017$ | $0{,}737 \pm 0{,}018$ |
| ResNet50 | MCLoss | $0{,}752 \pm 0{,}019$ | $0{,}752 \pm 0{,}020$ | $0{,}752 \pm 0{,}025$ |

Tabla 5: Comparación de los modelos ResNet y ResNet-BS sobre el problema Sorghum-100.

Como se puede apreciar en la Tabla 5, se logra compensar la degradación del rendimiento que producen las modificaciones realizadas en la arquitectura, a la vez que se reduce la cantidad de parámetros y FLOPS. Así, el modelo ResNet50-BS ofrece un rendimiento similar al obtenido con la arquitectura ResNet-50 en la Sección 5.3, con una disminución del 39.21 % en

la cantidad de parámetros y del 37.20 % en la cantidad de FLOPS. A pesar de que en un principio podría parecer que el desempeño de ResNet50-BS es inferior al de ResNet50 al utilizar la función de costos MCLoss, se realizó una prueba estadística t-test [49] para comparar los rendimientos de ambos modelos con dicha función de costos. Los resultados confirmaron que ambos modelos tienen un rendimiento similar con un nivel de confianza del 95 %.

## Análisis de tiempo de ejecución en sistemas embebidos

Para medir el impacto de las modificaciones se ejecutaron las distintas arquitecturas sobre los dispositivos embebidos Jetson Nano y Jetson Xavier NX típicamente utilizados para la ejecución de modelos de inteligencia artificial. La placa Jetson Nano tiene un procesador ARM Cortex-A57 de cuatro núcleos a 1.43 GHz, una placa de video Maxwell de 128 núcleos a 921 MHz y un peso de 141 gramos[43]. Por otro lado, la placa Jetson Xavier NX tiene un procesador NVIDIA Carmel ARMv8.2 de seis núcleos a 1.9 GHz, una placa de video Volta con 384 núcleos a 1100 MHz y un peso de 172 gramos[44]. En ambos casos, utilizamos el software del sistema JetPack 4.6[6].

Para medir los tiempos de ejecución en ambas placas, se utilizó la herramienta TensorRT[7]. Esta herramienta es ampliamente utilizada en la industria para acelerar los tiempos de inferencia de modelos convolucionales en unidades de cómputo GPU. De esta forma, se midió la cantidad de milisegundos que necesita cada modelo para realizar una inferencia sobre una imagen. Antes de llevar a cabo las mediciones se realizó un pre-calentamiento, o *warm up*, que consiste en realizar 3 inferencias con el objetivo de reducir el impacto de ciertos factores externos sobre las mediciones, como pueden ser el estado de la memoria cache de los procesadores. La Tabla 6 muestra los resultados obtenidos para los dispositivos Jetson Nano y Jerson Xavier NX. Se reportan el promedio de 10 valores medidos para cada modelo.

En la segunda columna de la Tabla 6 se muestran los resultados obtenidos en la placa Jetson Nano donde se logró una reducción del 35.68 % en el tiempo de inferencia para ResNet18-BS y del 21.50 % para ResNet50-BS en comparación con sus versiones originales. Por su parte, en tercera columna de la Tabla 6 se presentan los resultados sobre la placa Jetson Xavier NX donde se obtuvieron mejoras similares del 32.65 % y 18.40 % para ResNet18-BS y ResNet50-BS respectivamente. Para visualizar mejor el impacto de estas modificaciones, se presentan las Figuras 28a y 28b donde se analiza

[6] `https://developer.nvidia.com/embedded/jetpack`

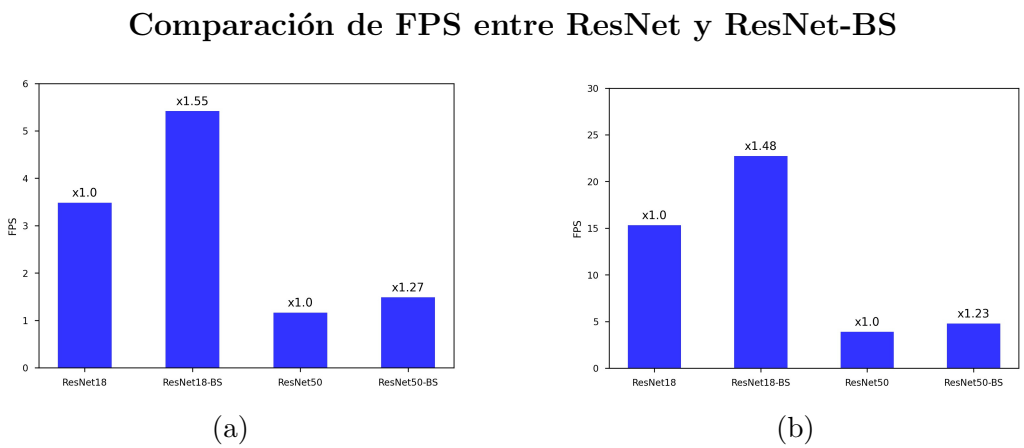
[7] `https://developer.nvidia.com/tensorrt`

| Modelo | Latencia Jetson Nano | Latencia Jetson Xavier NX |
|---|---|---|
| ResNet18 | 286,862$ms$ | 65,287$ms$ |
| ResNet18-BS | 184,505$ms$ | 43,967$ms$ |
| ResNet50 | 857,791$ms$ | 255,631$ms$ |
| ResNet50-BS | 673,089$ms$ | 208,574$ms$ |

Tabla 6: Latencia en milisegundos obtenidas en una Jetson Nano y una Jetson Xavier NX para los modelos ResNet y ResNet-BS.

la cantidad de imágenes procesadas por segundo (FPS) que se obtienen con cada modelo. La mayor ganancia se logró con la arquitectura ResNet18-BS con un incremento del 55 % en la Jetson Nano y del 48 % en la Jetson Xavier NX.

En conclusión, las modificaciones realizadas derivan en una mejora significativa en la velocidad de inferencia; lo cual es especialmente útil en aplicaciones donde se requiere una alta eficiencia en tiempo real o se cuenta con escasos recursos de computo.




(a) (b)

Figura 28: Impacto de las modificaciones realizadas a la arquitectura ResNet sobre la placa Jetson Nano(a) y Jetson Xavier NX(b). El eje de abscisas contiene los distintos modelos mientras que el eje de ordenadas muestra la cantidad de imágenes procesadas por segundo(FPS) que alcanzan estas arquitecturas.

# 6 Conclusiones y trabajos a futuro

En esta tesina se experimentó con redes convolucionales profundas, convoluciones BSC y funciones de costo para resolver el problema de clasificación de variedades de sorgo, haciendo foco en la eficiencia de los modelos. Se comparó al desempeño de las arquitecturas ResNet18 y ResNet50 con sus homólogos en ResNet-BS y se trato de disminuir las degradaciones en el rendimiento inducidas por las modificaciones propuestas. Finalmente, se midió la latencia de estos modelos sobre sistemas embebidos típicamente utilizados para la ejecución de aplicaciones de inteligencia artificial.

## 6.1. Conclusiones

En función de los experimentos presentados podemos concluir que el modelo ResNet50-BS aumenta aproximadamente un 25 % la cantidad de imágenes procesadas por segundo en sistemas embebidos como Jetson Nano o Jetson Xavier NX con respecto al modelo estándar ResNet-50. Sin embargo, para lograr mantener la misma capacidad de reconocimiento original fue necesaria la utilización de la función de costos MCLoss. También se debe notar que el modelo ResNet18-BS introduce un incremento de aproximadamente el 50 % en la cantidad de FPS, por lo que podría ser una buena opción a considerar en otras problemáticas donde la latencia del modelo sea un factor crucial.

Dentro de los experimentos realizados se puede observar que los mejores resultados fueron obtenidos utilizando el modelo ResNet-50 con la función de costos MCLoss. Con esta combinación alcanzó una precisión de 0.752 que supera los resultados presentados en [33] al utilizar el mismo modelo.

## 6.2. Trabajos a futuro

Como se observó en la sección de experimentación, las modificaciones introducidas para el proceso de entrenamiento tiene un mayor impacto que los cambios en la arquitectura. Es por esto, que consideramos que valdría la pena experimentar con otras funciones de costos, como TDSA[50], para evaluar su desempeño sobre nuestro problema. Sobre esta línea de pensamiento también creemos que sería muy provechoso, aunque también costoso, la implementación de algún método de ajuste de hiperparametros sobre las variables de la función de costos MCLoss ($\lambda$, $\mu$ y $\xi$).

Por otro lado, se podría experimentar con técnicas de entrenamiento no supervisado para pre-entrenar los modelos. En este trabajo debido a limitaciones en el poder de cómputo se utilizó como conjunto de pre-entrenamiento cifar10. Sin embargo, en [51] se muestra que el pre-entrenamiento de la red puede jugar un papel clave en los resultados obtenidos. Por este motivo consideramos que sería interesante analizar si métodos de entrenamiento no supervisado o un pre-entrenamiento en un conjunto de imágenes de plantas, como Herbarium[52], ayudan al modelo a obtener mejores resultados.

Finalmente, un objetivo más ambicioso podría ser analizar si el uso de la función de costos MCLoss permite entender mejor cuales son las zonas altamente discriminantes de una imagen dada. Los *feature maps* resultantes de la capa intermedia tomada para aplicar dicha función de costos están alineados con las distintas clases del problema y suelen contener información específica asociada a las distintas zonas discriminantes. En [12] se muestra cómo es posible visualizar las distintas zonas en una imagen específica. Por un lado, estas visualizaciones podrían brindar al usuario mayor información sobre las clasificaciones realizadas por la red. Por otro lado, existen trabajos, como [37], donde el foco se encuentra en localizar dichas zonas de la imagen que podrían beneficiarse de esta metodología.

# Referencias


[1] N. Ketkar y E. Santana. «Deep learning with Python». En: vol. 1. Springer, 2017, págs. 3-25.

[2] T. M. Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997, págs. 2-4.

[3] T. M. Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997, págs. 81-127.

[4] A. F. Agarap. «Deep learning using rectified linear units (relu)». En: *arXiv preprint arXiv:1803.08375* (2018).

[5] Y. LeCun y C. Cortes. «MNIST handwritten digit database». En: (2010). URL: `http://yann.lecun.com/exdb/mnist/`.

[6] K. O'Shea y R. Nash. «An introduction to convolutional neural networks». En: *arXiv preprint arXiv:1511.08458* (2015).

[7] Y. LeCun, Y. Bengio et al. «Convolutional networks for images, speech, and time series». En: *The handbook of brain theory and neural networks* 3361.10 (1995), pág. 1995.

[8] A. Krizhevsky, I. Sutskever y G. E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». En: *Advances in Neural Information Processing Systems*. Ed. por F. Pereira, C. Burges, L. Bottou y K. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[9] S. Ioffe y C. Szegedy. «Batch normalization: Accelerating deep network training by reducing internal covariate shift». En: *International conference on machine learning*. PMLR. 2015, págs. 448-456.

[10] N. Ketkar y E. Santana. «Deep learning with Python». En: vol. 1. Springer, 2017, págs. 260-262.

[11] D. Haase y M. Amthor. «Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets». En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, págs. 14600-14609.

[12] D. Chang, Y. Ding, J. Xie, A. K. Bhunia, X. Li, Z. Ma, M. Wu, J. Guo e Y.-Z. Song. «The devil is in the channels: Mutual-channel loss for fine-grained image classification». En: *IEEE Transactions on Image Processing* 29 (2020), págs. 4683-4695.

[13] X.-S. Wei, J. Wu, Q. Cui, O. M. Aodha, Y.-Z. Song, P. Yuxin, J. Tang, J. Yang y S. Belongie. «Deep learning for fine-grained image analysis: A survey». En: *arXiv preprint arXiv:1907.03069* (2019).

[14] S. Anwar, N. Barnes y L. Petersson. «A systematic evaluation: Fine-grained CNN vs. Traditional CNN classifiers». En: *arXiv preprint arXiv:2003.11154* (2020).

[15] P.-Y. Chou, C.-H. Lin y W.-C. Kao. «A Novel Plug-in Module for Fine-Grained Visual Classification». En: *arXiv preprint arXiv:2202.03822* (2022).

[16] Y. Chai, V. Lempitsky y A. Zisserman. «Symbiotic Segmentation and Part Localization for Fine-Grained Categorization». En: *2013 IEEE International Conference on Computer Vision*. 2013, págs. 321-328.

[17] J. Krause, H. Jin, J. Yang y L. Fei-Fei. «Fine-grained recognition without part annotations». En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, págs. 5546-5555.

[18] A. Dubey, O. Gupta, P. Guo, R. Raskar, R. Farrell y N. Naik. «Pairwise confusion for fine-grained visual classification». En: *Proceedings of the European conference on computer vision (ECCV)*. 2018, págs. 70-86.

[19] P. Hu, S. C. Chapman, X. Wang, A. Potgieter, T. Duan, D. Jordan, Y. Guo y B. Zheng. «Estimation of plant height using a high throughput phenotyping platform based on unmanned aerial vehicle and self-calibration: example for sorghum breeding». En: *European Journal of Agronomy* 95 (2018), págs. 24-32.

[20] L. Malambo, S. C. Popescu, S. C. Murray, E. Putman, N. A. Pugh, D. W. Horne, G. Richardson, R. Sheridan, W. L. Rooney, R. Avant et al. «Multitemporal field-based plant height estimation using 3D point clouds generated from small unmanned aerial systems high-resolution imagery». En: *International Journal of Applied Earth Observation and Geoinformation* 64 (2018), págs. 31-42.

[21] N. A. Pugh, D. W. Horne, S. C. Murray, G. Carvalho Jr, L. Malambo, J. Jung, A. Chang, M. Maeda, S. Popescu, T. Chu et al. «Temporal estimates of crop growth in sorghum and maize breeding enabled by unmanned aerial systems». En: *The Plant Phenome Journal* 1.1 (2018), págs. 1-10.

[22] F. Gnädinger y U. Schmidhalter. «Digital counts of maize plants by unmanned aerial vehicles (UAVs)». En: *Remote sensing* 9.6 (2017), pág. 544.

[23] K. Yamamoto, W. Guo, Y. Yoshioka y S. Ninomiya. «On plant detection of intact tomato fruits using image analysis and machine learning methods». En: *Sensors* 14.7 (2014), págs. 12191-12206.

[24] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez y C. McCool. «Deepfruits: A fruit detection system using deep neural networks». En: *sensors* 16.8 (2016), pág. 1222.

[25] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart y C. Stachniss. «UAV-based crop and weed classification for smart farming». En: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, págs. 3024-3031.

[26] S. P. Mohanty, D. P. Hughes y M. Salathé. «Using deep learning for image-based plant disease detection». En: *Frontiers in plant science* 7 (2016), pág. 1419.

[27] J. Ubbens, M. Cieslak, P. Prusinkiewicz e I. Stavness. «The use of plant models in deep learning: an application to leaf counting in rosette plants». En: *Plant methods* 14 (2018), págs. 1-10.

[28] A. Singh, B. Ganapathysubramanian, A. K. Singh y S. Sarkar. «Machine learning for high-throughput stress phenotyping in plants». En: *Trends in plant science* 21.2 (2016), págs. 110-124.

[29] L. Malambo, S. Popescu, N.-W. Ku, W. Rooney, T. Zhou y S. Moore. «A deep learning semantic segmentation-based approach for field-level sorghum panicle counting». En: *Remote Sensing* 11.24 (2019), pág. 2939.

[30] C. Gonzalo-Martın, A. Garcıa-Pedrero y M. Lillo-Saavedra. «Improving deep learning sorghum head detection through test time augmentation». En: *Computers and Electronics in Agriculture* 186 (2021), pág. 106179.

[31] K. He, X. Zhang, S. Ren y J. Sun. «Deep residual learning for image recognition». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.

[32] S. Ghosal, B. Zheng, S. C. Chapman, A. B. Potgieter, D. R. Jordan, X. Wang, A. K. Singh, A. Singh, M. Hirafuji, S. Ninomiya et al. «A weakly supervised deep learning framework for sorghum head detection and counting». En: *Plant Phenomics* (2019).

[33] C. Ren, J. Dulay, G. Rolwes, D. Pauli, N. Shakoor y A. Stylianou. «Multi-resolution outlier pooling for sorghum classification». En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, págs. 2931-2939.

[34] L. Herranz, S. Jiang y X. Li. «Scene recognition with cnns: objects, scales and dataset bias». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, págs. 571-579.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al. «Imagenet large scale visual recognition challenge». En: *International journal of computer vision* 115 (2015), págs. 211-252.

[36] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba y A. Oliva. «Learning deep features for scene recognition using places database». En: *Advances in neural information processing systems* 27 (2014).

[37] A. Stylianou, R. Pless, N. Shakoor y T. Mockler. «Classification and Visualization of Genotype x Phenotype Interactions in Biomass Sorghum». En: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, págs. 1352-1361.

[38] A. Krizhevsky, G. Hinton et al. «Learning multiple layers of features from tiny images». En: (2009).

[39] M. Burnette, R. Kooper, J. Maloney, G. S. Rohde, J. A. Terstriep, C. Willis, N. Fahlgren, T. Mockler, M. Newcomb, V. Sagan et al. «TERRA-REF data processing infrastructure». En: *Proceedings of the Practice and Experience on Advanced Research Computing*. 2018.

[40] R. Wightman, H. Touvron y H. Jégou. «Resnet strikes back: An improved training procedure in timm». En: *arXiv preprint arXiv:2110.00476* (2021).

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei. «Imagenet: A large-scale hierarchical image database». En: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, págs. 248-255.

[42] A. A. Süzen, B. Duman y B. Şen. «Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn». En: *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE. 2020, págs. 1-5.

[43] *Data sheet Nvidia Jetson Nano System-on-Module.* Nvidia. URL: https://developer.nvidia.com/downloads/embedded/dlc/jetson-nano-system-module-datasheet.

[44] *NVIDIA JETSON XAVIER NX. XAVIER PERFORMANCE. NANO SIZE.* Nvidia. URL: https://siliconhighway.com/wp-content/gallery/jetson-xavier-nx-datasheet-us-1154975-r4-web-original.pdf.

[45] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman y K. Zuiderveld. «Adaptive histogram equalization and its variations». En: *Computer vision, graphics, and image processing* 39.3 (1987), págs. 355-368.

[46] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin y A. A. Kalinin. «Albumentations: fast and flexible image augmentations». En: *Information* 11.2 (2020), pág. 125.

[47] G. Sun, H. Cholakkal, S. Khan, F. Khan y L. Shao. «Fine-grained recognition: Accounting for subtle differences between similar classes». En: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, págs. 12047-12054.

[48] S. Zhang, R. Du, D. Chang, Z. Ma y J. Guo. «Knowledge Transfer Based Fine-Grained Visual Classification». En: *2021 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2021, págs. 1-6.

[49] T. M. Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997, págs. 145-152.

[50] D. Chang, Y. Zheng, Z. Ma, R. Du y K. Liang. «Fine-grained visual classification via simultaneously learning of multi-regional multi-grained features». En: *arXiv preprint arXiv:2102.00367* (2021).

[51] R. Güldenring y L. Nalpantidis. «Self-supervised contrastive learning on agricultural images». En: *Computers and Electronics in Agriculture* 191 (2021), pág. 106510.

[52] R. de Lutio, J. Y. Park, K. A. Watson, S. D'Aronco, J. D. Wegner, J. J. Wieringa, M. Tulig, R. L. Pyle, T. J. Gallaher, G. Brown et al. «The Herbarium 2021 Half–Earth Challenge Dataset and Machine Learning Competition». En: *Frontiers in Plant Science* (2022), pág. 3320.