UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELECTRICA

EVALUACIÓN DE REDES DE NEURONALES CONVOLUCIONALES PRE-ENTRENADAS BAJO DIFERENTES ÍNDICES DE VARIABILIDAD DE COLOR

Tesis para optar al título de Ingeniero en Computación

ALEJANDRO GARCIA MAGAÑA

Asesor de Tesis: Dr. Jaime Cerda Jacobo

Morelia, Michoacan
03/2022

Agradecimientos

Gracias a mis padres Miguel Ángel García y Ma. Teresa Magaña por ser los principales promotores de mis sueños, gracias a ellos por cada día confiar y creer en mí y en mis expectativas, gracias a mi madre por estar dispuesta a acompañarme cada larga y agotadora noche de estudio, agotadoras noches en las que su compañía y la llegada de sus cafés era para mí como agua en el desierto; gracias a mi padre por siempre desear y anhelar siempre lo mejor para mi vida, gracias por cada consejo y por cada una de sus palabras que me guiaron durante mi vida.

Gracias a mi pareja, Cristina Ortiz, por entenderme en todo, gracias a ella porque en todo momento fue un apoyo incondicional en mi vida, fue la felicidad encajada en una sola persona.

A lo largo de mi estancia en la Facultad de Ingeniería eléctrica de la UMSNH, he tenido la fortuna de conocer excelentes profesores, compañeros y amigos de los cuales siempre quedaran los buenos recuerdos y las grandes experiencias vividas a quienes brindo mis más sinceros agradecimientos.

A mi asesor el Dr. Jaime Cerda Jacobo , por su tiempo ,orientación y consejos brindados a lo largo de el desarrollo de este proyecto

A mí querida casa de estudio UMSNH por brindarme el espacio para culminar mis estudios profesionales.

Dedicatoria

A mis padres Miguel Ángel Garcia y Ma. Teresa Magaña , muchos de mis logros se los debo a ellos por su apoyo que siempre he tenido, por su cariño ,todas las enseñanzas y todo el tiempo que han dedicado pacientemente a mi formación como ser humano.Sobre todo por ser un excelente ejemplo de vida a seguir.

A mis hermanos Miguel Ángel Garcia , José Luis Garcia, Ana Lilia Garcia , cuyo ejemplo de esfuerzo y trabajo me inspiraron a cumplir mis metas.

Así como toda mi familia y amigos que me han motivado a concluir mis metas.

Resumen

Las Redes Neuronales Convolucionales (CNN) son muy utilizadas en la actualidad en trabajos de clasificación de imágenes , por lo que la eficiencia es clave al momento de implementar.
El trabajo de tesis consiste en demostrar utilizando redes neuronales convolucionales, se puede ver afectado al rendimiento al clasificar imágenes si se utilizan imágenes con diferentes niveles de variabilidad del color, así como también con imágenes muy luminosas.Para lograr esto primero se hizo la estructura y los principales componentes que interfieren en el rendimiento de una red neuronal convolucional ,posteriormente se utiliza el lenguaje Python donde se implementa una función que calcula el índice de variabilidad de color y la luminosidad de una imagen dada. Una vez teniendo el índice de variabilidad de color se divide el conjunto de imágenes original en subconjuntos con altos y bajos niveles de este índice, posterior se hace uso de la biblioteca Keras para la implementación de la red neuronal convolucional,se utiliza una red pre-entrenada donde esta se entrena con los diferentes subconjuntos de imágenes para así obtener el rendimiento con cada uno de los conjuntos, finalmente se compara estos resultados en busca de cual subconjunto tuvo un mejor rendimiento y si esto es debido a los índices de variabilidad de color.

Palabras Clave: Función, Modulo, Redes Neuronales, Píxel

Abstract

Convolutional Neural Networks (CNN) are currently widely used in image classification work, so efficiency is key when implementing.
The thesis work consists of demonstrating, using convolutional neural networks, performance can be affected if images with different levels of color variability are used, as well as with very bright images. To achieve this, the structure and the main components were first made. that interfere in the performance of a convolutional neural network, later the Python language is used where a function is implemented that calculates the index of color variability and the luminosity of a given image. Once having the color variability index, the original set of images is divided into subsets with high and low levels of this index, later the Keras library is used for the implementation of the convolutional neural network, a preliminary network is used. trained where it is trained with the different subsets of images in order to obtain the performance with each of the sets, finally these results are compared in search of which subset had a better performance and if this is due to the color variability indices.

Glosario de terminos

- **CNN :** Red Neuronal Convolucional
- **XOR :** Exclusive Or
- **ImageNet :** ImageNet es el conjunto de datos por excelencia en la actualidad para evaluar algoritmos de clasificación, localización y reconocimiento de imágenes
- **RNCP :** Redes Neuronales Convolucionales Pre-entrenadas
- **Phyton :** Lenguaje de programación interpretado multiparadigma
- **CPU :** Unidad Central de Procesamiento
- **GPU :** Unidad de procesamiento gráfico

Índice de figuras

Índice de cuadros

Índice general

1 Introducción

1.1. Antecedentes

La historia de las redes neuronales [5], [6], [14] comienza en el estudio de el cerebro como una forma de ver la computación (1936,Alan Turing),para después comenzar en 1949 se desarrollo una regla de aprendizaje (Donald Hebb). Comenzaría el desarrollo de la red neuronal mas antigua llamada perceptron (1957,Frank Rosenblatt) esta fue diseñado con el propósito de reconocimiento de imágenes ,aunque era un gran avance esta era incapaz de reconocer diferentes tipos de patrones como la compuerta lógica XOR (Exclusive Or).Fue hasta 1980 cuando Kunihiko Fukushima desarrollo un módulo para el reconocimiento de patrones visuales, fue hasta entonces que se descubrió que al usar mas capas en el modelo del perceptron y utilizando la técnica llamada retropropagacion, se tenían mejores resultados (LeCun et al., 1989).

Esta red fue diseñada para combatir la variabilidad de figuras 2D [1] y reconocía patrones directamente de imágenes con un mínimo pre-procesamiento, estaba enfocada en el reconocimiento de letras manuscritas
El siguiente sucesor fue la Red Neuronal Convolucional (LeCun et al., 1998b), a diferencia de el anterior en el que sólo los pesos de conexión de la primera capa podían ser modificados, actualiza todos los pesos de conexión y usa retropropagación para el entrenamiento. Podemos ver su estructura en la figura 1.1.

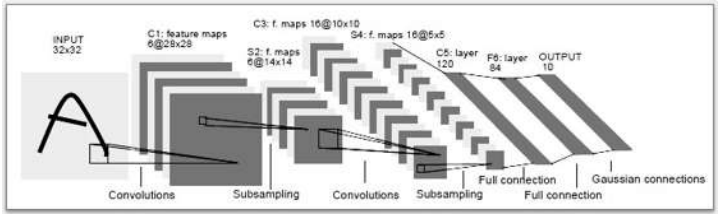

Figura 1.1: Modelo (Lecun et al.,1998)

AlexNet, Krizhevsky et al, tuvieron un gran impacto en el campo del aprendizaje automático, específicamente en la aplicación del aprendizaje profundo a la visión artificial. Ganó la famosa competencia ImageNet LSVRC-2012 de 2012 por un amplio margen (15,3 % VS 26,2 % (segundo lugar) tasas de error). La red tenía una arquitectura muy

similar a la de LeNet-5 pero con más capas y filtros, lo que resultaba en una red más grande y más parámetros para aprender.

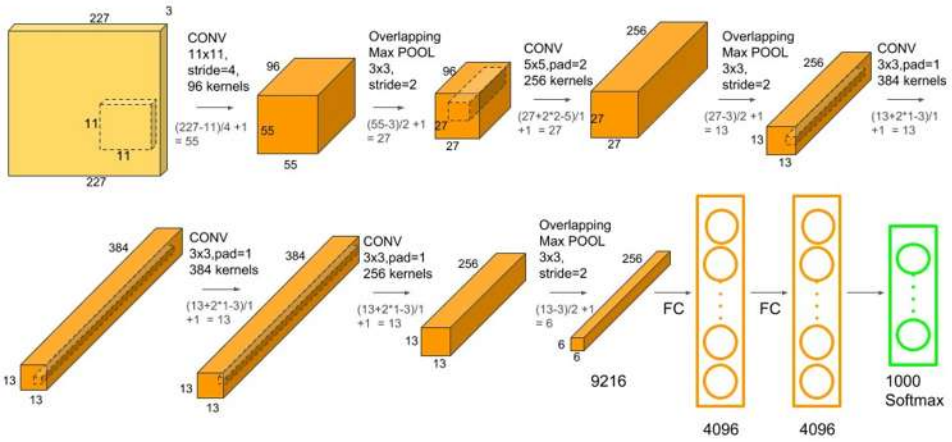

Figura 1.2: AlexNet 2012

VGG es un modelo de red neuronal convolucional propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford en 2014. El modelo alcanza una precisión del 92,7 % en ImageNet, que es un conjunto de datos de más de 14 millones de imágenes que pertenecen a 1000 clases. Es conocido por su diseño uniforme y ha tenido mucho éxito en muchos dominios, como vemos en la imagen 1.3.

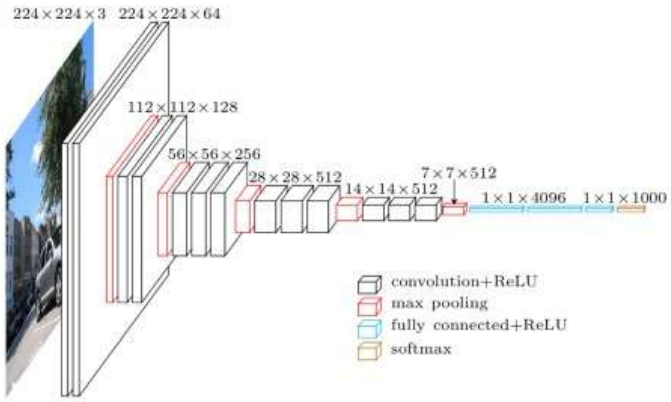

Figura 1.3: VggNet 2014

Algunos de los avances más impresionantes en inteligencia artificial en los últimos años se han producido en el campo del aprendizaje profundo. La traducción del lenguaje natural, el reconocimiento de imágenes y el juego son tareas en las que los modelos de aprendizaje profundo se han acercado o incluso superado el rendimiento a nivel humano.

A través de su poder y escalabilidad, las redes neuronales se han convertido en el modelo definitorio del aprendizaje profundo. Las redes neuronales están compuestas por neuronas, donde cada neurona individualmente realiza solo un cálculo simple. El poder de una red neuronal proviene en cambio de la complejidad de las conexiones que pueden formar estas neuronas.

1.2. Objetivo General

Las redes convencionales es lo mas usado en la actualidad en tareas de clasificación de imágenes y reconocimiento de patrones por fácil que es utilizarla y por obtener tan buen rendimiento cuando son entrenadas con un conjunto de imágenes suficientemente grande.

En este trabajo de tesis se realiza la evaluación de redes convolucionales preentrenadas a través de conjuntos de imágenes con diferentes índices de variabilidad de color, para demostrar que al usar imágenes con este índice de variabilidad en el entrenamiento afecta al rendimiento de una red convolucional en la clasificación de imágenes.

1.3. Objetivos Particulares

- Estudio de las Redes Neuronales, arquitectura, y sus principales características que influyen en el rendimiento al momento de su implementación.
- Diseño de algoritmos en lenguaje Python para la obtención de el índice de variabilidad de color ,desviación estándar y luminosidad sobre una imagen.
- Dividir en 2 subconjuntos de igual tamaño con alto y bajo índice para la desviación estándar,luminosidad y variabilidad de color.
- Utilizar la biblioteca keras para la implementación de Redes neuronales Pre-entrenadas.
- Utilizar servicios de el servidor Kaggle para el trabajo de computo así mismo como realizar la evaluación sobre rendimiento de redes neuronales convolucionales.
- Implementar las redes neuronales convolucionales pre-entrenadas en el servidor Kaggle.
- Entrenamiento de la red neuronal convolucional en kaggle con los diferentes subconjuntos.

- Comparación del rendimiento obtenido en clasificación de imágenes en las diferentes redes neuronales entrenadas.
- Disminuir el índice de variabilidad en imágenes con alto índice.

1.4. Justificación

El estudio de las redes neuronales desde su base nos ayuda a entender el comportamiento que estas tienen así como la forma en que son capaces de detectar características en imágenes para su clasificación, conociendo su arquitectura se comprende de que manera afecta en el rendimiento las imágenes de entrenamiento, así, aumentando o reduciendo su eficiencia.

Se utiliza el lenguaje de programación Python ya que desde su lanzamiento en 1991 ha tenido un crecimiento asombroso. La legibilidad del código desarrollado en Python es sencillo, elegante y busca ser consistente. Esto permite que la estructura del lenguaje se asemeje a la estructura que implementamos los seres humanos y a lo que conocemos como lenguaje matemático permitiendo que este sea leído como un pseudocódigo.Las librerías de Python son amplias. Existen miles de librerías de data science y matemáticas, pero el sistema de empaquetamiento de este lenguaje permite construir librerías nuevas sobre las ya existentes para que estas sean más amplias y potentes.

El diseño de nuestro algoritmo se hizo para poder asociar una imagen a un numero que nos dirá que tanta variabilidad de color, desviación estándar o luminosidad tiene esa imagen, para posteriormente dividirlas en subconjuntos con altos y bajos niveles de este índice.

Keras una biblioteca de Redes Neuronales de Código Abierto que está especialmente diseñada para posibilitar la experimentación en poco tiempo con redes de Aprendizaje Profundo ofreciéndonos herramientas para facilitar la construcción de CNN.

Se utilizo el servidor Kaggle para el trabajo de computo ya que nos ofrece de manera gratuita el uso de su GPU para ejecutar nuestros programas de una manera mas rápida con un mayor recurso computacional, realizar pruebas y en este caso realizar el computo para nuestra red neuronal convolucional.

El diseño se hizo con las mejores 4 CNN pre-entrenadas, ya que entrenar una red neuronal convolucional como veremos son miles de parámetros que se deben ajustar, y si lo hiciéramos desde cero, para poder obtener buenos resultados requeriría de que esta tuviera un entrenamiento de semanas e incluso meses, por cuestiones de tiempo utilizaremos las redes pre-entrenadas que estas ya tienen inicializados estos valores para su uso en clasificación de imágenes.

Se entreno la red con diferentes conjuntos de imágenes de acuerdo a el índice de variabilidad de color, desviación estándar y luminosidad para poder medir la eficiencia

de el modelo y comparar con las demás pruebas. Así como fomentar una nueva área de estudio para las CNN donde esta pueda ser integrada para mejorar la eficiencia.

1.5. Metodología

Será de gran utilidad poder tener una referencia de que tanto afecta una imagen muy luminosa o con cambios de colores a una CNN en la predicción de nuevas imágenes, entonces, primero se realiza una investigación acerca de el funcionamiento de las Redes neuronales y las CNN además de conocer las capas por las que estas se conforman. Por otra parte, se implementa el algoritmo que obtiene el índice de variabilidad de color sobre imágenes, posterior mente se implementara código Python para la creación de un modelo convolucional para este entrenarlo con los diferentes conjuntos de datos, para así poder evaluar el desempeño con los conjuntos de imágenes con índices de variabilidad de color, desviación estándar, luminosidad alto y bajo que tienen las imágenes, para de esta manera ver si estos factores afectan al rendimiento de nuestra red neuronal convolucional.

1.6. Descripción de los capítulos

En el capitulo 1, se presenta la introducción del tema de tesis, el motivo de la investigación, un poco de los antecedentes. También se muestra los inicios de las Redes Neuronales que después dieron paso a las Redes Neuronales Convolucionales.

En el capitulo 2, se presenta las Redes Neuronales, Redes Neuronales Profundas, Redes Neuronales Convolucionales, así como las partes importantes que las conforman.

En el capitulo 3, se presenta el diseño de las RNCP (Redes Neuronales Convolucionales Pre-entrenadas).

En el capitulo 4, se encuentra el desarrollo de la evaluación y el diseño de los distintos algoritmos utilizados durante el trabajo de tesis.

En el capitulo 5, se presenta los resultados de la evaluación de las RNCP, se analizan y se buscan patrones de comportamiento en base a las imágenes usadas para el entrenamiento.

En el capitulo 6, se presenta las conclusiones sobre el tema, además de algunas sugerencias para trabajos futuros en base a los resultados obtenidos.

2 Redes neuronales artificiales y aprendizaje profundo

2.1. Neurona artificial

Las neuronas artificiales están inspiradas en la neurona biológica la cual recibe señales de entrada a través de sinapsis de otras neuronas que se conectan a las dendritas de la neurona [5]. La información recibida se procesa en una señal de salida a través de el axón, la cual es la multiplicación entre la señal de entrada y la fuerza de la sinapsis que se conecta a la dendrita, en modelos computacionales las señales de entrada son los datos de entrada y la señal procesada equivale a la unidad de salida como se muestra en la figura 2.1 y 2.2.

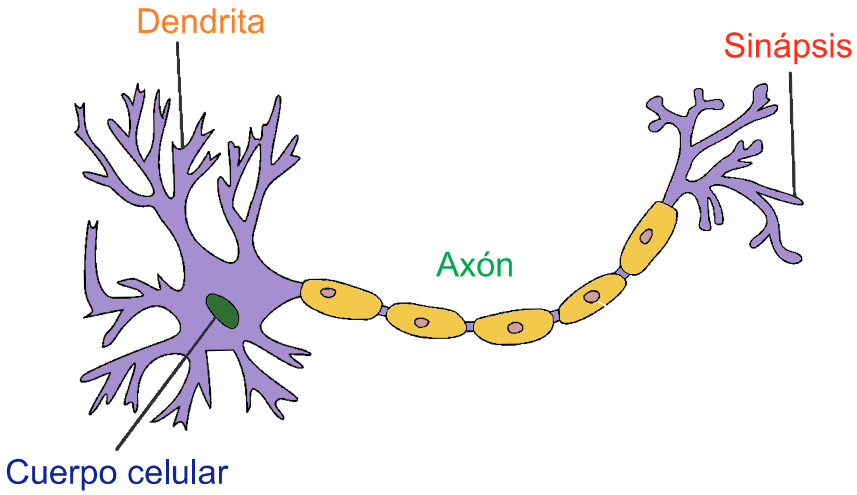

Figura 2.1: Neurona biológica

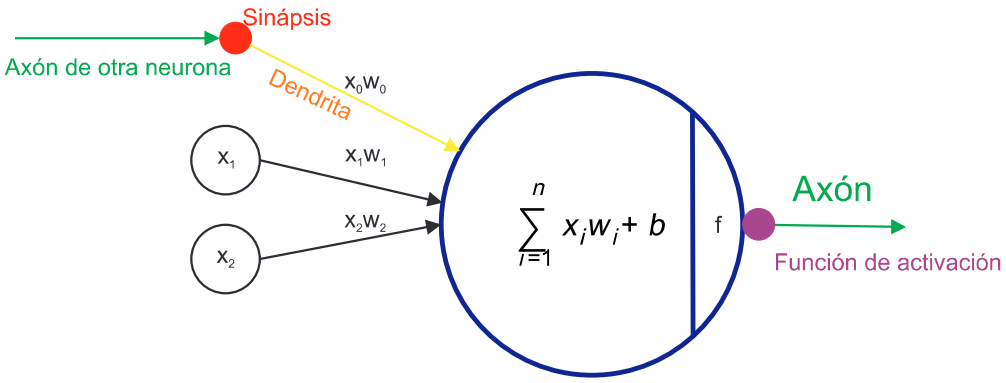

Figura 2.2: Neurona artificial

2.1.1. Unidad Lineal

La unidad lineal es el componente fundamental de una red neuronal: la neurona individual [5], [6]. Como diagrama, una neurona (o unidad) con una entrada se ve como la figura 2.3.

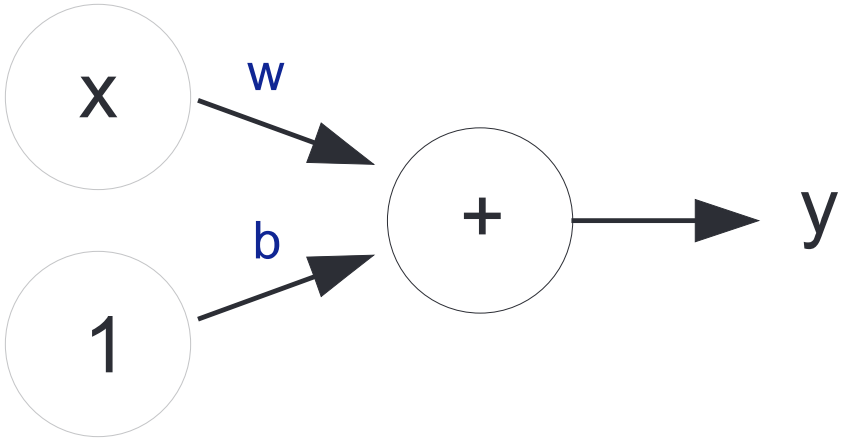

Figura 2.3: Unidad Lineal $y = wx + b$

La entrada es x. Su conexión con la neurona tiene un peso que es w. Siempre que un valor fluya a través de una conexión, se multiplica el valor por el peso de la conexión. Para la entrada x, lo que llega a la neurona es $w * x$. Una red neuronal aprende modificando sus pesos.

La b es un tipo especial de ponderación que se llamara en este proyecto "sesgo". El sesgo no tiene ningún dato de entrada asociado; en cambio, se pone un 1 en el diagrama para que el valor que llegue a la neurona sea simplemente b (ya que $1 * b = b$). El sesgo

permite a la neurona modificar la salida independientemente de sus entradas.

2.1.2. Entradas Múltiples

Se puede simplemente agregar más conexiones de entrada a la neurona [2], [5], una para cada característica adicional. Para encontrar la salida, se multiplica cada entrada por su peso de conexión y luego se suman todas.

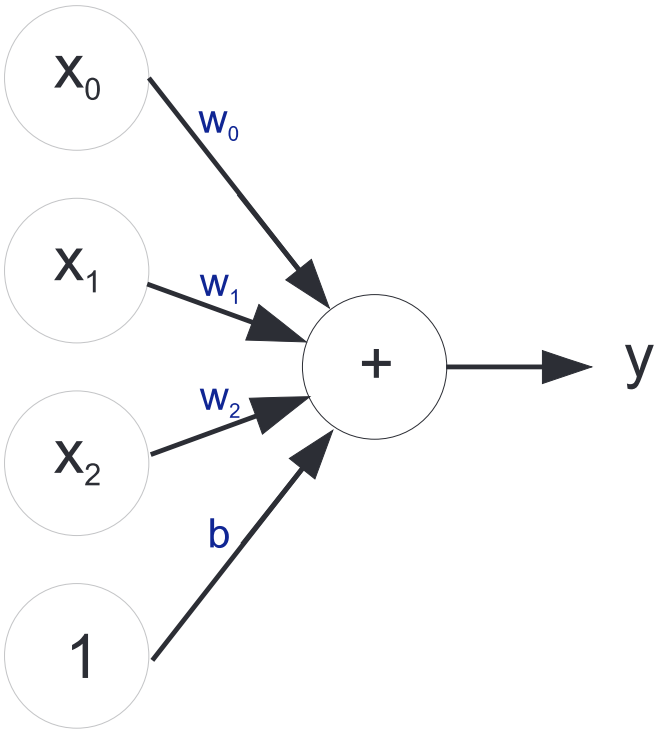

Figura 2.4: Una unidad lineal con 3 entradas

La fórmula para esta neurona de la figura 2.4 es $y = w_0x_0 + w_1x_1 + w_2x_2 + b$. Una unidad lineal con dos entradas encajará en un plano, y una unidad con más entradas encajará en un hiper plano.

2.2. Redes Neuronales Profundas

La idea clave aquí es la modularidad, la construcción de una red compleja a partir de unidades funcionales más simples.

2.2.1. Capas

Las redes neuronales suelen organizar sus neuronas en capas. Cuando se agrupan unidades lineales que tienen un conjunto común de entradas, se obtiene una capa densa.

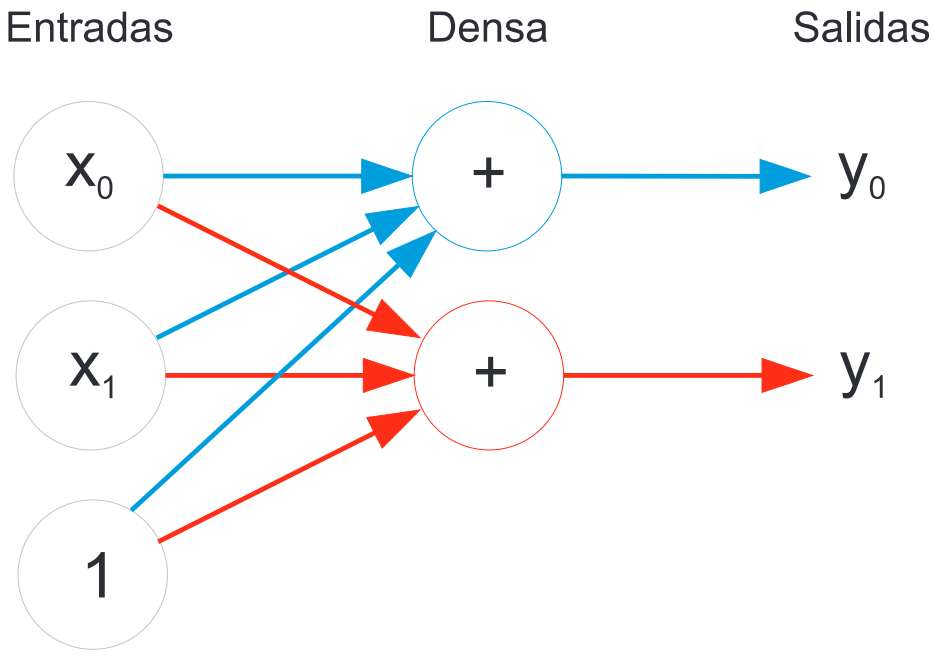

Figura 2.5: Una capa densa de dos unidades lineales que reciben dos entradas y un sesgo.

A través de una pila profunda de capas, una red neuronal puede transformar sus entradas de formas cada vez más complejas. En una red neuronal bien entrenada, cada capa es una transformación que se acerca un poco más a una solución.

2.2.2. Función de Activación

Sin embargo, resulta que dos capas densas sin nada en el medio no son mejores que una sola capa densa por sí misma. Las capas densas por sí mismas nunca pueden salir del mundo de las líneas y los planos [5], [6].

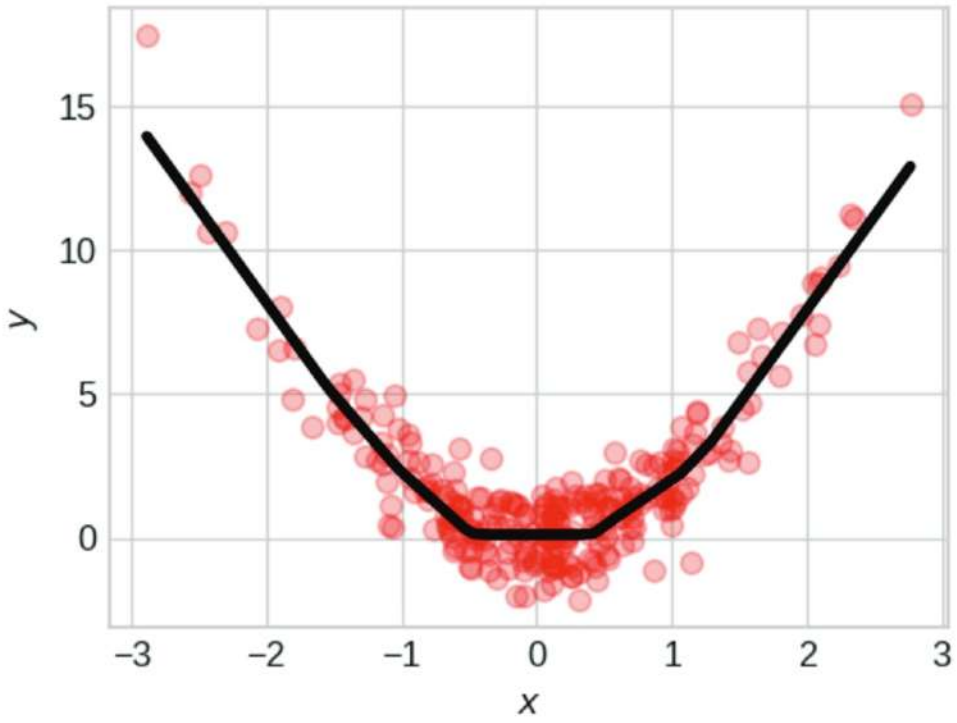

Figura 2.6: Función de activación

Sin funciones de activación, las redes neuronales solo pueden aprender relaciones lineales. Para ajustar las curvas, se necesita usar funciones de activación.

Las funciones de activación producen la salida y se consideran como la función de decisión, entre las principales se encuentran.

Función identidad

$$f(x) = x, \forall x \tag{2.1}$$

Esta función $f(x) = y$ devuelve el mismo resultado que la entrada, esta generalmente se usa en el la ultima capa de la red neuronal.

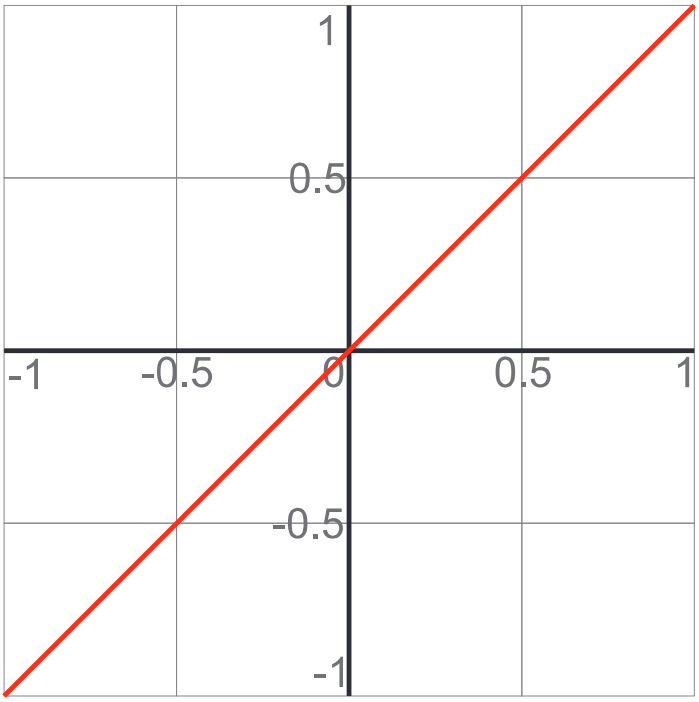

Figura 2.7: Función Identidad

Función escalón unitario binario

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.2}$$

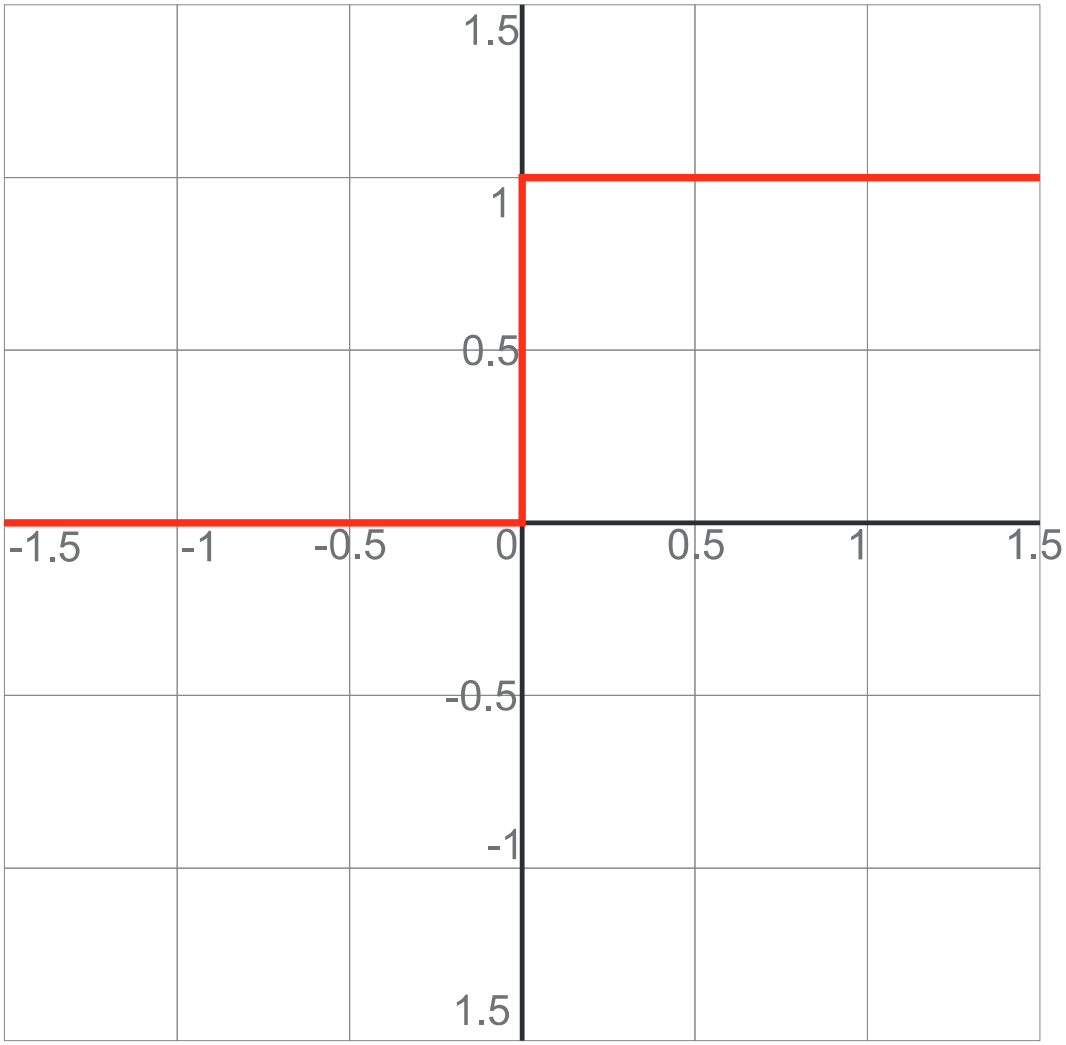

Figura 2.8: Función escalón unitario binario

Esta es una función que devuelve un resultado binario que solo puede tomar 2 valores $(0,1)$, en base a un umbral θ.

Función escalón unitario bipolar

$$f(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.3}$$

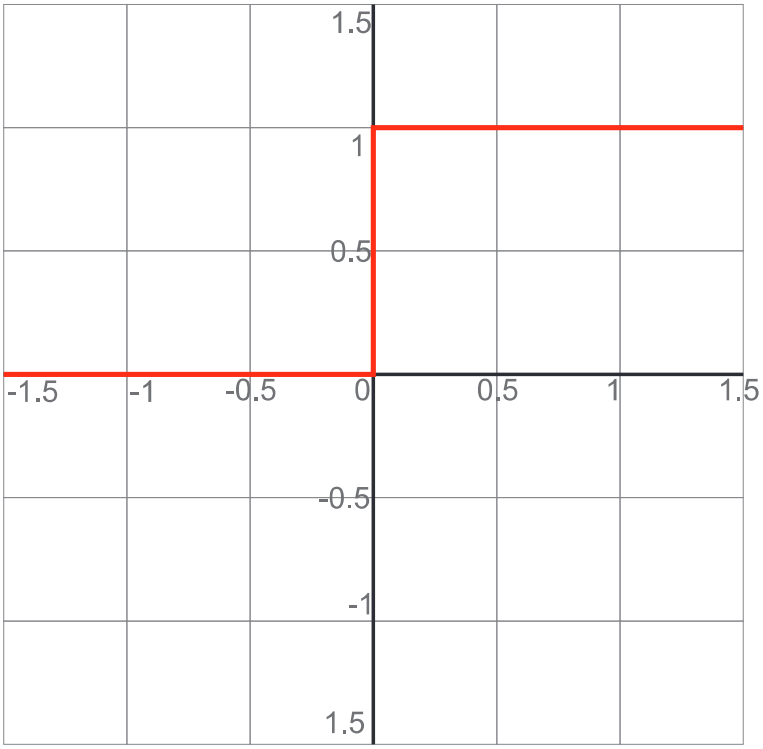

Figura 2.9: Función escalón unitario bipolar

Por igual que la función de escalón unitario binario esta devuelve únicamente dos valores posibles (-1,1) en base a un umbral θ.

Función sigmoide binaria

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

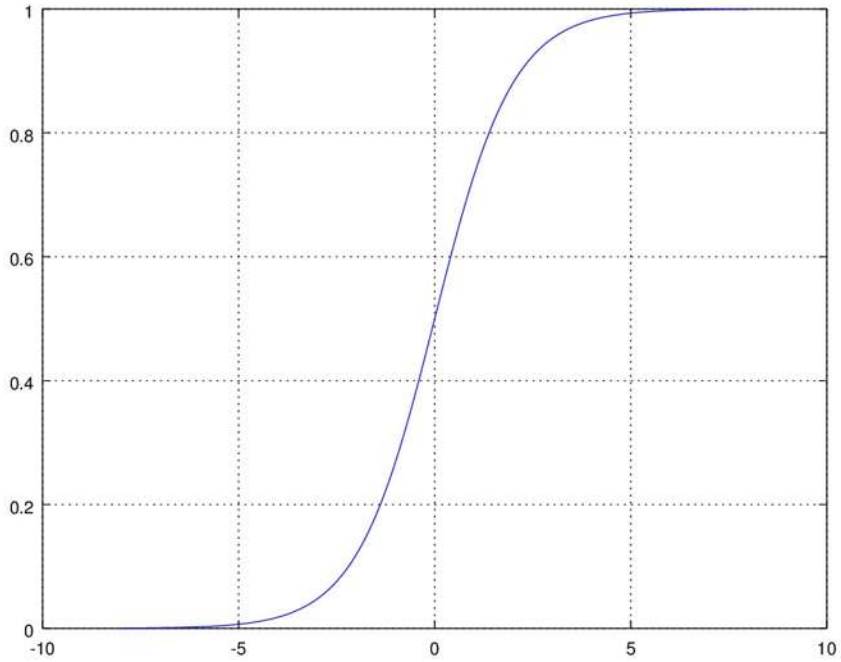

Figura 2.10: Función sigmoide binario bipolar

Esta función devuelve un resultado que va en el rango [0,1]. Una ventaja que esta presenta es que es diferenciable (se puede derivar), por lo que esta característica es

crucial para el desarrollo del algoritmo de retropropagacion, su derivada se calcula de la forma siguiente.

$$
\begin{aligned}
f'(x) &= \frac{\partial}{\partial x}(\frac{1}{1+e^{-x}}) \\
f'(x) &= \frac{e^{-x}}{1+e^{-x}}(\frac{1}{1+e^{-x}}) \\
f'(x) &= \frac{e^{-x}}{1+e^{-x}}(\frac{1}{1+e^{-x}}) \\
f'(x) &= \frac{e^{-x}}{(1+e^{-x})^2} \\
f'(x) &= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
f'(x) &= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
f'(x) &= \frac{1}{(1+e^{-x})} - (\frac{1}{1+e^{-x}})^2 \\
f'(x) &= f(x) - f(x)^2 \\
f'(x) &= f(x)(1-f(x))
\end{aligned}
\tag{2.5}
$$

La función sigmoide fue la función de activación mas usada para el entrenamiento de redes neuronales en retropropagacion, ya no es recomendable usarla ya que presenta el uso de grandes recursos computacionales en comparación de las otras funciones de activación.

Función tangente hiperbólica

$$
f(x) = \frac{2}{1+\exp^{-2x}} - 1 \tag{2.6}
$$

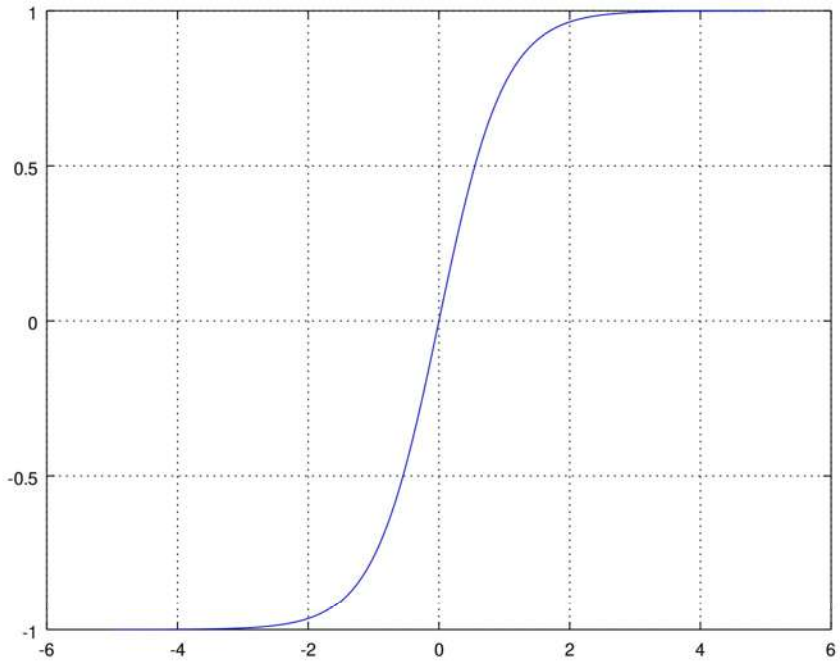

Figura 2.11: Función tangente hiperbólica

También es conocido como función sigmoide bipolar es una función similar a la sigmoide binaria pero esta produce salida en el rango[-1,1].

ReLu (Rectified Linear Unit)

$$f(x) = max(0, x) \tag{2.7}$$

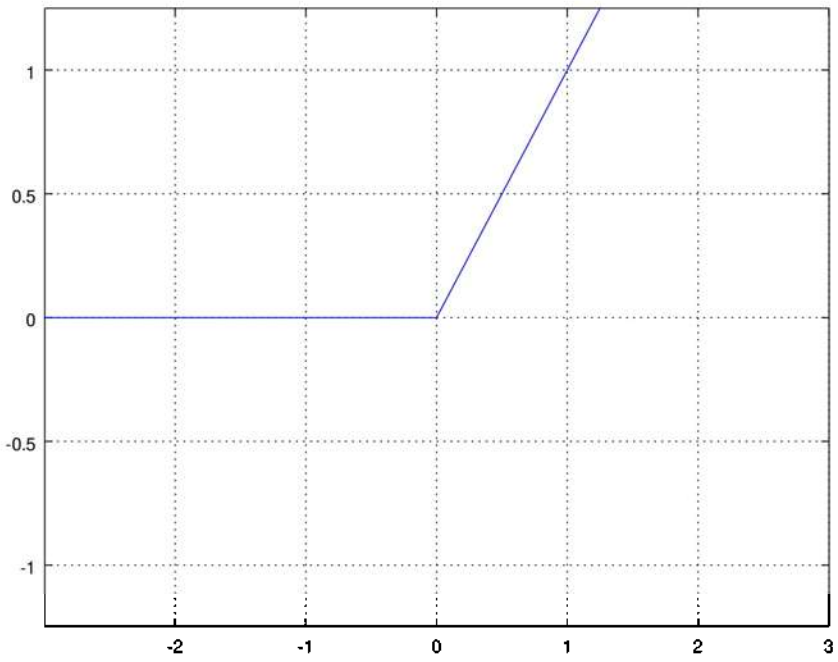

Figura 2.12: Función tangente hiperbólica

Devuelve un resultado en el rango [0,∞]

ReLu es la función de activación mas utilizada en el mundo en este momento y se utiliza en casi todas las redes neuronales convolucionales o el aprendizaje profundo y esta es usada en las capas ocultas de nuestra red neuronal, no en la de salida.

Se puede ver en el gráfico que cuando el valor es mayor a 0 la salida es 1, y cuando el valor es menor o igual a 0 la salida es 0, por lo tanto la derivada se define como:

$$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.8}$$

Softmax

La función de activación softmax convierte una lista de valores en probabilidades que van desde 0 a 1.

La formula de de la función softmax es.

$$f_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.9}$$

Softmax es utilizado en los problemas de clasificación múltiple. Se observa en la figura 2.2 un diagrama de la operación de softmax, en un sistema donde recibe 3 entradas y así mismo devuelve 3 salidas.

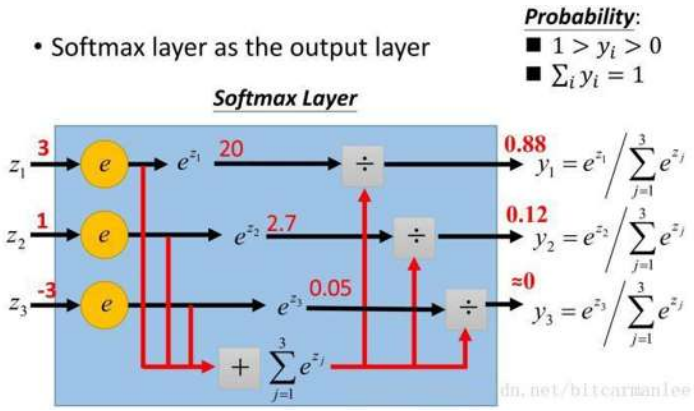

Figura 2.13: Función Softmax

En la función de activación Softmax, la suma de todas las salidas y, es igual a 1 ya que se trata de probabilidades.

$$\sum_{i=1}^{n} y_i = 1 \tag{2.10}$$

2.2.3. Función de perdida

La función de pérdida mide la disparidad entre el valor real del objetivo y el valor que predice el modelo [5], [6].

Error absoluto medio MAE

Una función de pérdida común para los problemas de regresión es el error absoluto medio o MAE. Para cada predicción y_{pred}, MAE mide la disparidad del objetivo verdadero y_{true} por una diferencia absoluta $|y_{true} - y_{pred}|$.

La pérdida total de MAE en un conjunto de datos es la media de todas estas diferencias absolutas.

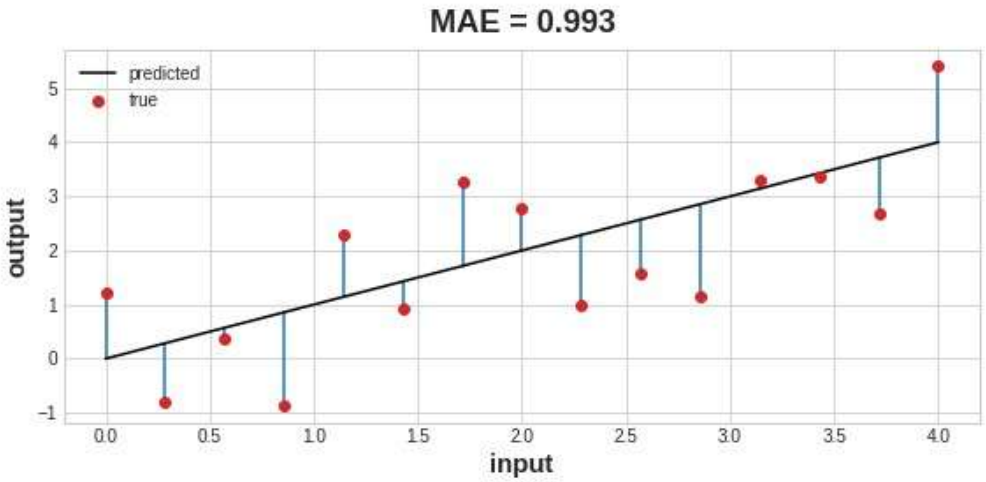

Figura 2.14: Error absoluto medio

Pérdida de entropía cruzada

Se usa esta función de pérdida de entropía cruzada(categorical cross entropy) cuando hay dos o más clases de etiquetas.

Los valores de etiqueta de las categorías m utilizadas por su red de entrenamiento deben ser vectores vectorizados m-dimensionales, donde un índice del vector es 1, y los índices restantes son 0, correspondientes a una categoría. De esta manera, las categorías se vectorizan, correspondientes a los m valores de probabilidad entrenados por la red neuronal, y el vector correspondiente a la salida con el mayor valor de probabilidad es la etiqueta que representa.

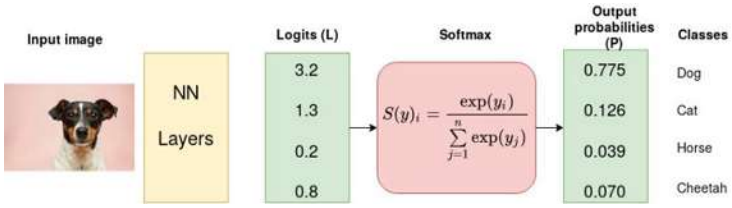

Figura 2.15: Diagrama de clasificación

Esta función de perdida toma la salida de la función de activación Softmax, ya que estamos hablando de clasificación de 2 o mas clases como lo es la figura 2.15.

Una vez tomada la salida de la función de activación este compara ambos vectores, las probabilidades y los valores de verdad y así medir la distancia de ambos valores.

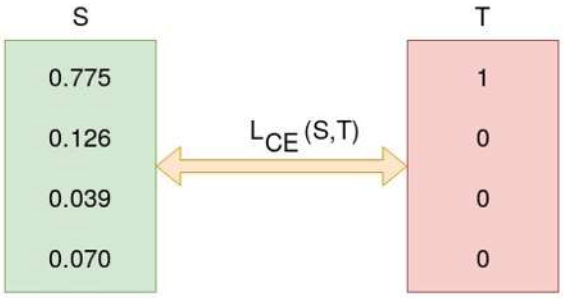

Figura 2.16: diagrama de entropía cruzada

El objetivo es hacer que la salida del modelo sea lo más cercana posible a la salida deseada (valores de verdad).

2.2.4. El optimizador

El optimizador es un algoritmo que ajusta los pesos de la red neuronal para minimizar la pérdida.

Gradiente estocástico descendente

El gradiente es el conjunto de todas las derivadas parciales de una función. En el caso de machine learning, es el gradiente de la función de coste.

Prácticamente todos los algoritmos de optimización utilizados en el aprendizaje profundo pertenecen a una familia llamada descenso de gradiente estocástico. Son algoritmos iterativos que entrenan una red en pasos. Un paso de entrenamiento es el siguiente.

1. Se prueban algunos datos de entrenamiento y se ejecutan a través de la red para hacer predicciones.
2. Se mide la pérdida entre las predicciones y los valores reales.
3. Finalmente, se realizan los ajustes a los pesos en una dirección que reduzca la pérdida.

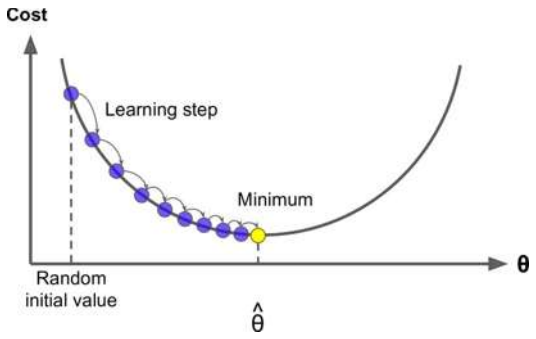

Figura 2.17: Gradiente descendente

La muestra de datos de entrenamiento de cada iteración se denomina minibatch, mientras que una ronda completa de datos de entrenamiento se denomina época. La cantidad de épocas para las que entrena es la cantidad de veces que la red verá cada ejemplo de entrenamiento como se muestra en la figura 2.17.

Optimizador Adam

Adam es un gradiente estocástico descendiente que combina estrategia de aprendizaje adaptativo en momentos de primer orden y segundo orden.

El algoritmo de Adam es diferente del descenso de gradiente estocástico tradicional. El descenso de gradiente estocástico mantiene una tasa de aprendizaje única (es decir, alfa) para actualizar todos los pesos, y la tasa de aprendizaje no cambia durante el proceso de entrenamiento. Y Adam calcula la estimación de momento de primer orden gradiente y estimación de momento de segundo orden. El autor del algoritmo Adam para diseñar tasas de aprendizaje adaptativo independientes para diferentes parámetros lo describe como un conjunto de ventajas de dos descensos de gradiente estocástico extendidos, a saber:

1. El algoritmo de gradiente adaptativo (AdaGrad) reserva una tasa de aprendizaje para cada parámetro para mejorar el rendimiento en gradientes dispersos (es decir, problemas de lenguaje natural y visión por computadora).
2. Propagación cuadrática media (RMSProp) basada en el gradiente de peso media de magnitud más cercana . La tasa de aprendizaje se reserva adaptativamente para cada parámetro. Esto significa que el algoritmo tiene un excelente rendimiento en problemas no estacionarios y en línea.

El algoritmo de Adam obtiene las ventajas de los algoritmos AdaGrad y RMSProp. Adam no solo calcula la tasa de aprendizaje de parámetros adaptativos en función del valor medio del primer momento como el algoritmo RMSProp, sino que también hace un uso completo del valor medio del segundo momento del gradiente (es decir, la varianza no centrada).

2.2.5. Tasa de aprendizaje y tamaño de lote

La tasa de aprendizaje y el tamaño de los minibatches son los dos parámetros que tienen el mayor efecto sobre cómo se desarrolla el entrenamiento de gradiente estocástico descendente.

De manera intuitiva, dado que la tasa de aprendizaje escala la derivada, determina cuánto se actualiza cada parámetro después de cada iteración de descenso de gradiente.

Encontrar el tamaño correcto de la tasa de aprendizaje depende del proyecto y los datos proporcionados.

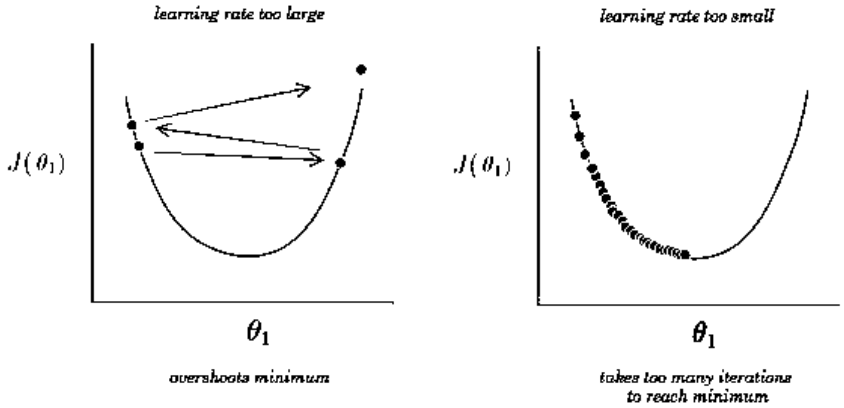

Figura 2.18: Gradiente descendente

Una tasa de aprendizaje menor significa que la red necesita ver más minibatches antes de que sus pesos converjan a sus mejores valores, por el contrario una tasa de aprendizaje muy alta, los cambios serán muy grandes y será difícil encontrar los coeficientes que minimicen la función de coste, como se observa en la figura 2.18.

Las redes neuronales se entrenan en una serie de épocas. Cada época consta de un pase hacia adelante y un pase de propagación hacia atrás sobre todas las muestras de entrenamiento proporcionadas. Ingenuamente, podemos calcular el gradiente verdadero calculando el valor del gradiente de cada caso de entrenamiento de forma independiente, luego sumando los vectores resultantes. Esto se conoce como aprendizaje por Full batch y proporciona una respuesta exacta a la pregunta de qué dirección de paso es óptima, en lo que respecta al descenso de gradiente.

Alternativamente, podemos optar por actualizar los pesos de entrenamiento varias veces en el transcurso de una sola época. En este caso, ya no estamos calculando el gradiente verdadero; en lugar de eso, estamos calculando una aproximación del gradiente real, utilizando sin embargo muchas muestras de entrenamiento incluidas en cada división de la época. Esto se conoce como aprendizaje por mini lotes.

2.3. Redes Neuronales Convolucionales

2.3.1. Vision General

Los parámetros de la capa convolucional consisten en un conjunto de filtros que se pueden aprender(matrices). Cada filtro es pequeño espacialmente (a lo largo de ancho y alto), pero se extiende a través de toda la profundidad del volumen de entrada [6], [10], [14].

Durante el pase hacia adelante, se desliza cada filtro a lo largo y ancho del volumen de entrada y calcula los productos escalares entre las entradas del filtro y la entrada en cualquier posición. A medida que se desliza el filtro sobre el ancho y la altura del volumen de entrada, esto produce un mapa de activación bidimensional que da las respuestas de ese filtro en cada posición espacial. Intuitivamente la red aprende filtros que se activan cuando ven algún tipo de característica visual como un borde de alguna orientación o una mancha de algún color en la primera capa, o eventualmente patrones enteros en forma de panal o rueda en capas superiores de la red.

Las redes neuronales convolucionales están formadas por neuronas que tienen pesos y sesgos que se pueden aprender. Cada neurona recibe algunas entradas, realiza un producto escalar y, opcionalmente, lo sigue con una no linealidad. Toda la red todavía expresa una única función de puntuación diferenciable: desde los píxeles de la imagen sin procesar en un extremo hasta las puntuaciones de la clase en el otro.

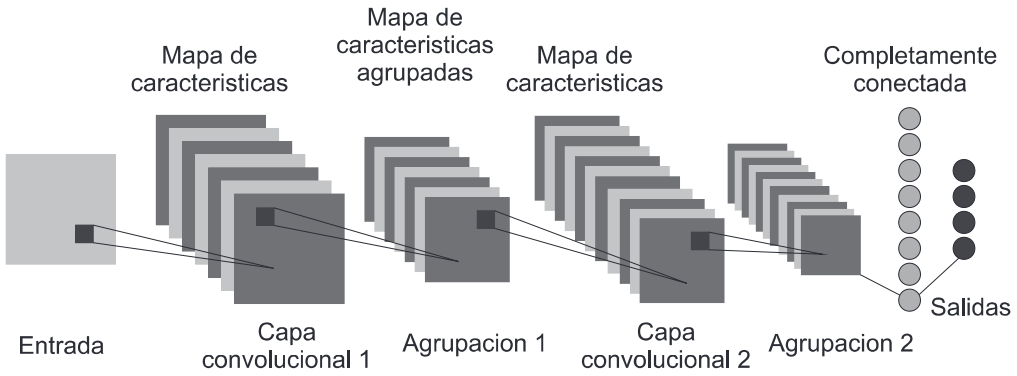

Figura 2.19: Capas de una red convolucional

Las arquitecturas CNN asumen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades en la arquitectura. Estos hacen que la función de reenvío sea más eficiente de implementar y reducen enormemente la cantidad de parámetros en la red.
Una CNN simple es una secuencia de capas, y cada capa transforma un volumen de activaciones en otro a través de una función diferenciable. Se utilizan tres tipos princi-

pales de capas a arquitecturas: Capa convolucional , Pooling de capa y capa totalmente conectado.Se agrupan estas capas para formar una arquitectura ConvNet completa.

- Entrada:[32x32x3] mantendrá los valores de píxeles sin procesar de la imagen, en este caso una imagen de ancho 32, alto 32 y con tres canales de color R, G, B.
- La capa Convolucional: calcula la salida de las neuronas que están conectadas a regiones locales en la entrada, cada una calculando un producto escalar entre sus pesos y una pequeña región a la que están conectadas en el volumen de entrada. Esto puede resultar en un volumen como [32x32x12] si se usa 12 filtros.
- La capa Relu: aplica una función de activación por elementos, como la max(0,x)con umbral en cero. Esto deja el tamaño del volumen sin cambios ([32x32x12]).
- La capa Pool: Realiza una operación de reducción de resolución a lo largo de las dimensiones espaciales (ancho, alto), dando como resultado un volumen como [16x16x12].
- La capa Completamente Conectada: calcula los puntajes de la clase, lo que dará como resultado un volumen de tamaño [1x1x10].

2.3.2. Convolución

Durante la propagación, existen en las CNN varios filtros (matrices) pequeños que van pasando por una parte de la imagen. La CNN aprende filtros (la cantidad es determinada por la profundidad de la capa) que se activan cuando se observa un borde o un contorno con cierta orientación especifica.

Cada filtro puede ser representado como una neurona de salida.

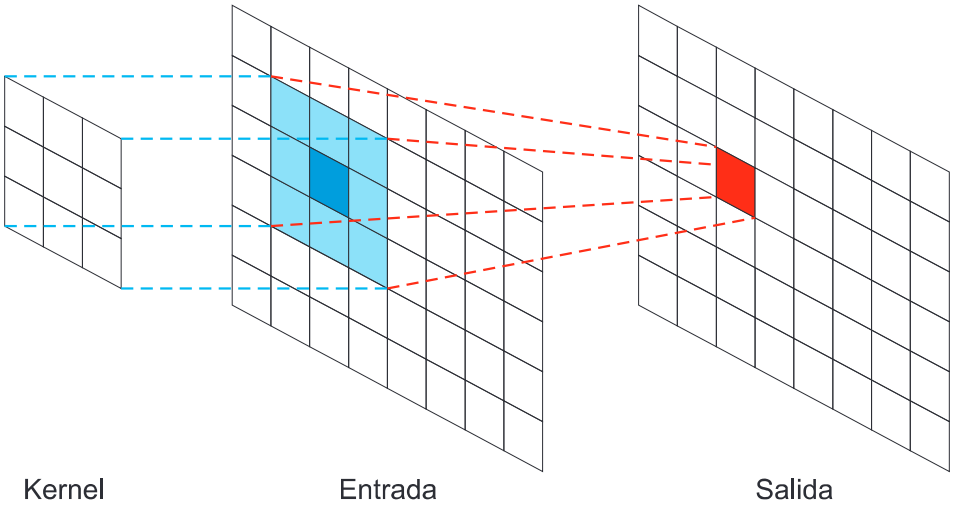

Figura 2.20: Operación Convolucional

Esta capa acepta un dato de entrada con el siguiente volumen:

$$W_1 x H_1 x D_1 \tag{2.11}$$

Y a la salida se produce el volumen:

$$W_2 x H_2 x D_2 \tag{2.12}$$

En el que:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \tag{2.13}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \tag{2.14}$$

$$D_2 = K \tag{2.15}$$

Donde:
W es el ancho (width)
H es la altura (height)
D es la profundidad (depth)
F tamaño de el filtro
S es el paso (stride)
P es la cantidad de zero padding
K es el numero de filtros (filters)

La ecuación de convolución generalizada es :

$$net_j^n = \sum_{k=1}^{K} x_k^{n-1} * w_{kj}^n + b^n \tag{2.16}$$

En el que : n es el numero de la capa
j es el numero del filtro de salida
K es la cantidad de filtros en la capa n-1 o n

2.3.3. Disposición Espacial

Una vez explicado la conectividad de cada neurona en la capa de conversión al volumen de entrada, pero aún se analiza cuántas neuronas hay en el volumen de salida o cómo están organizadas. Tres hiper parámetros controlan el tamaño del volumen de salida: la profundidad, el paso y el relleno de ceros .

1. Profundidad: la profundidad del volumen de salida es un hiper parámetro: corresponde a la cantidad de filtros que se desea usar, cada uno aprendiendo a buscar algo diferente en la entrada. Por ejemplo, si la primera capa convolucional toma

como entrada la imagen sin procesar, entonces diferentes neuronas a lo largo de la dimensión de profundidad pueden activarse en presencia de varios bordes orientados o manchas de color. Se refiere a un conjunto de neuronas que miran la misma región de la entrada como una columna de profundidad.

2. Paso: Se debe especificar el paso con el que se desliza el filtro. Cuando el paso es 1, se mueve los filtros un píxel a la vez. Cuando el paso es 2 (o excepcionalmente 3 o más, aunque esto es poco común en la práctica), los filtros saltan 2 píxeles a la vez a medida que se desplazan. Esto producirá volúmenes de salida más pequeños espacialmente.

3. Relleno de Ceros: El tamaño de este relleno de ceros es un hiper parámetro. La buena característica del relleno de cero es que permite controlar el tamaño espacial de los volúmenes de salida (lo más común es que, se usa para preservar exactamente el tamaño espacial del volumen de entrada, por lo que el ancho de entrada y salida y la altura son los mismos).

Se puede calcular el tamaño espacial del volumen de salida en función del tamaño del volumen de entrada (W), el tamaño del campo receptivo de las neuronas de la capa convolucional (F), el paso con la que se aplican (S) y la cantidad de relleno de cero utilizado (P).

La fórmula correcta para calcular cuántas neuronas "encajan" viene dada por:

$$\frac{W - F + 2P}{S} + 1 \tag{2.17}$$

Restricciones en los pasos

Se debe tener en cuenta de nuevo que los hiper parámetros de disposición espacial tienen restricciones mutuas. Por ejemplo, cuando la entrada tiene un tamaño $W = 10$,no se usa relleno de ceros $P = 0$ y el tamaño del filtro es $F = 3$, entonces sería imposible usar paso $S = 2$, ya que:

$$\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 2 * 0}{1} + 1 = 4{,}5 \tag{2.18}$$

Es decir no es un número entero, lo que indica que las neuronas no encajan de forma ordenada y simétrica en la entrada .

2.3.4. Capa de reducción de muestreo (Pooling)

Es común insertar periódicamente una capa de agrupación entre capas sucesivas de Conv en una arquitectura ConvNet. Su función es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y cálculos en la

red y, por lo tanto, controlar también el sobre ajuste. La capa de agrupación funciona de forma independiente en cada segmento de profundidad de la entrada y la redimenciona espacialmente, utilizando la operación MAX.

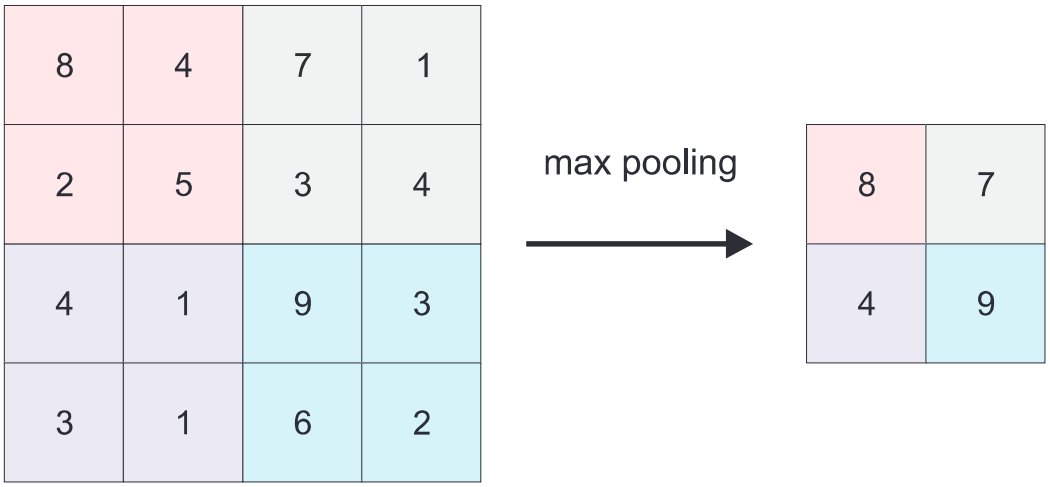

Figura 2.21: capa de reducción, pooling de tamaño 2x2

La forma más común es una capa de agrupación con filtros de tamaño $2x2$ aplicados con una zancada de 2 muestras descendentes de cada corte de profundidad en la entrada por 2 a lo largo de la anchura y la altura, descartando el 75 % de las activaciones. En este caso, cada operación MAX tomaría un máximo de 4 números (una pequeña región de $2x2$ en un segmento de profundidad, fig 2.21). La dimensión de profundidad permanece sin cambios. De manera más general, la capa de agrupación:

- Acepta un volumen de tamaño $W_1 H_1 D_1$
- Requiere dos hiper parámetros:
  - Su extensión espacial F
  - El paso S
- Produce un volumen de tamaño $W_2 H_2 D_2$
  - $W_2 = \frac{(W_1 - F)}{S} + 1$
  - $H_2 = \frac{(H_1 - F)}{S} + 1$
  - $D_2 = D_1$

En donde W es el ancho (width), H es la altura (height), D es la profundidad(depth), F es la extensión espacial (spatial extent) y S es el paso (stride).

2.3.5. Capa Completamente Conectada

Al final de una red neuronal convolucional, hay una o más capas totalmente conectadas (cuando dos capas están "totalmente conectadas", cada uno de los nodos de la primera capa está conectado a cada uno de los nodos de la segunda capa). Su trabajo es realizar una clasificación basada en los atributos extraídos mediante las convoluciones. Por lo general, la última capa totalmente conectada contiene una función de activación softmax, que arroja un valor de probabilidad de 0 a 1 para cada una de las etiquetas de clasificación que el modelo intenta predecir.

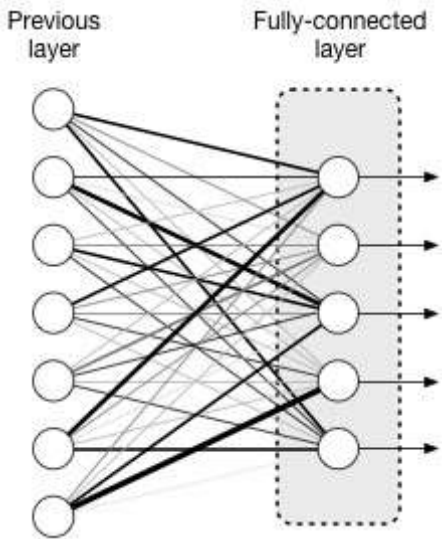

Figura 2.22: Capa completamente conectada

2.4. Luminancia en imágenes RGB

Dada una imagen en color representada en el modelo de color RGB **BrIma**, [7], [8] , esto significa que se tiene una imagen compuesta por píxeles de color, donde cada píxel tiene tres atributos numéricos que representan la intensidad (o luminosidad) de los primarios rojo, verde y azul, cuya mezcla representa la color de píxel correspondiente.

Se entiende que esta imagen RGB, esta compuesta por tres matrices con las mismas dimensiones que la imagen en color. Uno con el valor rojo de cada píxel en el píxel de la imagen en color (el canal rojo), otro con el verde (el canal verde) y otro con los valores azules (el canal azul).

Figura 2.23: Estructura de una imagen RGB

En este caso se busca el promedio de la imagen completa, se esperaría que ese valor se correlacione perceptualmente con la luminosidad de la imagen.

2.4.1. Luminosidad Relativa

La luminosidad relativa del color no es solo un concepto abstracto. Ha sido medido y estandarizado en función de cada longitud de onda de color puro: la función de luminosidad . La luminosidad es una información tan útil que el espacio de color XYZ describe los colores con coordenadas X, Y y Z, donde la coordenada Y es la luminosidad relativa del color, este sistema se sigue usando como referencia para definir los colores que percibe el ojo humano y otros espacios de color. Por ejemplo, si la luz verde y azul se emiten con la misma intensidad (técnicamente hablando), se percibe la luz verde como más luminosa que la azul.

En este caso particular, se desea medir el ruido de la imagen a través de la Desviación Estándar de los valores de los píxeles, y la percepción del ruido está claramente relacionada con los cambios en la luminosidad de la imagen. Por lo tanto, la conversión de los valores de los componentes de píxeles a su luminosidad relativa se adapta perfectamente a lo que se busca.
El NTSC RGB es un espacio de color RGB particular, con un estándar (Rec. 601) que establece desde 1982 cómo calcular el luma a partir de los valores del componente de color NTSC RGB. NTSC es un acrónimo de "National Television System Committeez también es el nombre de su estándar para la transmisión de televisión analógica.
La siguiente fórmula muestra cómo se deben calcular la luminosidad y la luminancia de acuerdo con NTSC (Rec. 601). En esta fórmula, Y significa luminosidad(o luminancia). Correspondientemente,*Red*, *Green* y *Blue*.

$$Y = 0{,}299 * Rojo + 0{,}587 * Verde + 0{,}114 * Azul \tag{2.19}$$

Los coeficientes en la fórmula 2.19 provienen del componente Y correspondiente a los primarios RGB $(Y_{rojo}, Y_{verde}, Y_{azul})$ que define el espacio de color. Estos coeficientes son los valores de Y normalizados para obtener su suma igual a 1.

Al aplicarse esta formula en una imagen RGB se obtiene una sola matriz que relaciona las matrices iniciales RGB en una única que nos indica la luminosidad de cada píxel.

3 Redes Neuronales Convolucionales Pre-entrenadas

En primer lugar, se definirá qué es ImageNet [4], [10]. Es una base de datos de imágenes que consta de 14.197.122 imágenes organizadas según la jerarquía de WordNet. esta es una iniciativa para ayudar a investigadores, estudiantes y otras personas en el campo de la investigación de la imagen y la visión.

ImageNet también organiza concursos, uno de los cuales fue ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), que desafió a los investigadores de todo el mundo a encontrar soluciones que produzcan las tasas de error más bajas entre los 1 y los 5 primeros (la tasa de error de los 5 primeros sería el porcentaje de imágenes en las que la etiqueta correcta no es una de las cinco etiquetas más probables del modelo). El concurso ofrece un conjunto de entrenamiento de 1.000 clases de 1,2 millones de imágenes, un conjunto de validación de 50 mil imágenes y un conjunto de prueba de 150 mil imágenes.

Un modelo pre-entrenado es un modelo que fue entrenado con un conjunto de datos de referencia para resolver un problema similar al que queremos abordar. Debido al coste computacional del entrenamiento de tales modelos, así como en la complejidad a la hora de elegir la arquitectura óptima, el Transfer Learning de modelos bien conocidos y precisos se ha convertido es una práctica común, hay varios modelos pre-entrenados de los cuales se explica los utilizados en este trabajo de tesis.

3.1. VGG19

La red pre-entrenada VGG19 es una CNN profunda que se usa para clasificar imágenes [12].
Su arquitectura esta dada por la tabla 3.1.

Nombre de capa	Definicion capa
conv1	(3x3,64,paso 1) x 2
maxpool	2x2 max pool, paso2
conv2	(3x3,128) x 2
maxpool	2x2 max pool
conv3	(3x3,256) x 4
maxpool	2x2 max pool
conv4	(3x3,512) x 4
maxpool	2x2 max pool
conv5	(3x3,512) x 4
maxpool	2x2 max pool
Fc	Fully Connected(4096)
Fc	Fully Connected(4096)
Fc	Fully Connected(1000)
	softmax

Cuadro 3.1: Arquitectura VGG19

3.2. ResNet50

En 2012 en el concurso de clasificación LSVRC2012 AlexNet ganó el primer premio, después de eso ResNet fue lo más interesante que le pasó a la visión por computadora y al mundo del aprendizaje profundo [15].

Su arquitectura esta dada por la tabla 3.2.

Nombre de capa	Definicion capa
conv1	7x7,64,paso 2
pooling	3x3 max pool, paso2
conv2 x	$\begin{pmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{pmatrix} x3$
conv3 x	$\begin{pmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{pmatrix} x3$
conv4 x	$\begin{pmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{pmatrix} x3$
conv5 x	$\begin{pmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{pmatrix} x3$
Fc	Fully Connected(1000)
	softmax

Cuadro 3.2: Arquitectura ResNet50

3.3. InceptionV3

El modelo Inception es un avance importante en el desarrollo de clasificadores de redes neuronales convolucionales (CNN). Tiene una arquitectura compleja (muy diseñada) y utiliza muchos trucos para impulsar el rendimiento en términos de velocidad y precisión [11].

Su arquitectura esta dada por la tabla 3.3.

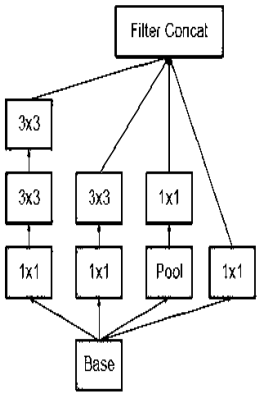

Figura 3.1: Modulo 1 inception

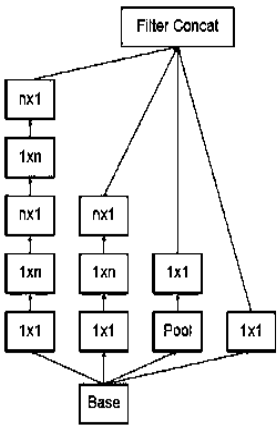

Figura 3.2: Modulo 2 inception

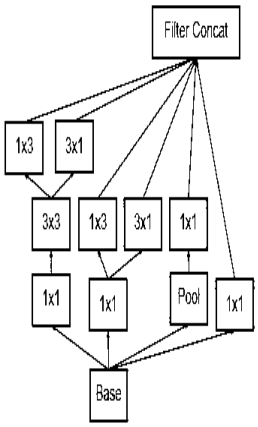

Figura 3.3: Modulo 3 inception

Nombre de capa	Definicion capa/paso
conv	3x3/2
conv	3x3/1
conv padded	3x3/1
pool	3x3/2
conv	3x3/1
conv	3x3/2
Inception	Figura 3.1 x 3
Inception	Figura 3.2 x 5
Inception	Figura 3.3 x 2
pool	8x8
linear	logits
	softmax

Cuadro 3.3: Arquitectura InceptionV3

3.4. Xception

Xception es una arquitectura de red neuronal convolucional profunda que involucra convoluciones separables en profundidad. Esta red fue presentada por Francois Chollet que trabaja en Google, Inc. [13].

Su arquitectura esta dada por la figura 3.4.

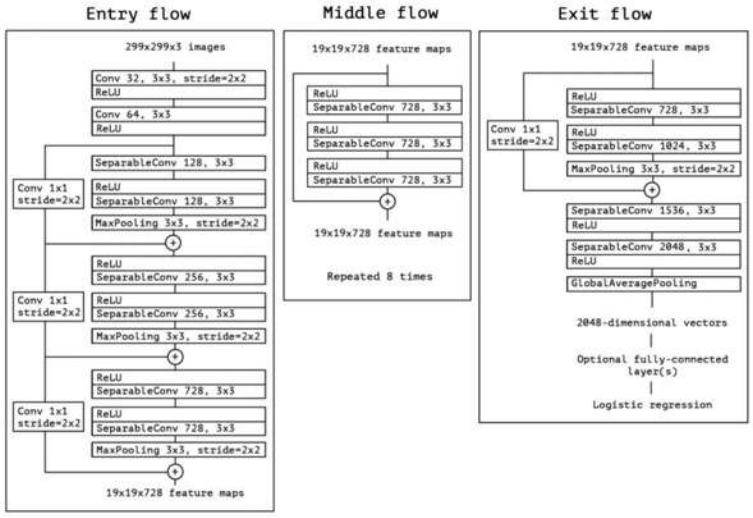

Figura 3.4: Arquitectura Xception

3.5. DenseNet

Una DenseNet es un tipo de red neuronal convolucional que utiliza conexiones densas entre capas, a través de bloques densos, donde conectamos todas las capas (con tamaños de mapas de características coincidentes) directamente entre sí. Para preservar la naturaleza de retroalimentación, cada capa obtiene entradas adicionales de todas las capas anteriores y pasa sus propios mapas de características a todas las capas posteriores [16].

Su arquitectura esta dada por la tabla 3.4.

Nombre de capa	Definicion capa/paso
Conv	7x7/2
Max pool	3x3/2
Dense block	$\begin{pmatrix} 1x1, conv \\ 3x3, conv \end{pmatrix} x6$
Transition Layer	$\begin{pmatrix} 1x1, conv \\ 2x2, averagepool/2 \end{pmatrix}$
Dense block	$\begin{pmatrix} 1x1, conv \\ 3x3, conv \end{pmatrix} x12$
Transition Layer	$\begin{pmatrix} 1x1, conv \\ 2x2, averagepool/2 \end{pmatrix}$
Dense block	$\begin{pmatrix} 1x1, conv \\ 3x3, conv \end{pmatrix} x48$
Transition Layer	$\begin{pmatrix} 1x1, conv \\ 2x2, averagepool/2 \end{pmatrix}$
Dense block	$\begin{pmatrix} 1x1, conv \\ 3x3, conv \end{pmatrix} x32$
Average pool	7x7
	softmax

Cuadro 3.4: Arquitectura DenseNet

3.6. EfficientNet-B0

La arquitectura EfficientNet-Bo no fue desarrollada por ingenieros sino por la propia red neuronal. Desarrollaron este modelo utilizando una búsqueda de arquitectura neuronal multiobjetivo que optimiza tanto la precisión como las operaciones de punto flotante [17].

Su arquitectura esta dada por la tabla 3.5.

Operador	Canales
Conv3x3	32
MBConv1,k3x3	16
MBConv6,k3x3	24
MBConv6,k5x5	40
MBConv6,k3x3	80
MBConv6,k5x5	112
MBConv6,k5x5	192
MBConv6,k3x3	320
Conv1x1,Pooling,FC	1280

Cuadro 3.5: Arquitectura EfficientNet-B0

4 Evaluación de las Redes Neuronales Convolucionales Pre-entrenadas

En este capitulo se presenta la metodología utilizada para la evaluación de las redes neuronales.

Para la realización de este proyecto se utilizaron diferentes herramientas para la evaluación de las redes neuronales convolucionales preentrenadas

4.1. Imágenes utilizadas

El conjunto de imágenes que se utilizo es sobre clasificación de flores, este se puede conseguir en kaggle , contiene un total de 4242 imágenes divididas en las clases siguientes:

- manzanilla tulipán
- rosa
- girasol
- diente de león

Para cada clase hay unas 800 fotos en promedio, por lo que se podría decir que tiene mas o menos la misma cantidad de fotos para cada clase como se observa en la figura 4.1 .

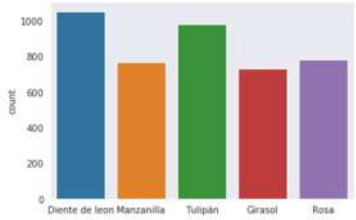

Figura 4.1: Cuenta de las clases

Se muestra una imagen de cada clase en la figura 4.2.

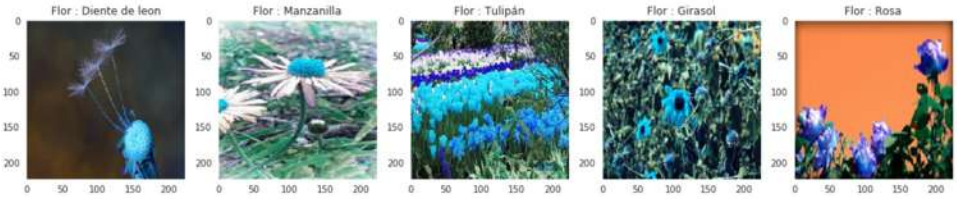

Figura 4.2: Demostración de las clases

Para la utilización de este conjunto de datos y su posterior evaluación se requirió dimensionar todas las imágenes, se utilizo una dimensión de 224x224 (alto y ancho), ya que esta resolución es necesaria para las redes neuronales que se utilizaran a continuación.

4.2. Bibliotecas de python

Python es un lenguaje de programación muy potente, además de ser de alto nivel, este tiene herramientas enfocadas al aprendizaje automático por lo que fueron necesarias las siguientes bibliotecas para la realización.

4.2.1. Keras

La biblioteca keras contiene todo lo necesario para la creación y evaluación de redes neuronales, además de ser sencilla de utilizar y contar con los modelos pre-entrenados que es el motivo de estudio de este trabajo.

4.2.2. Pandas

La biblioteca pandas esta enfocada en el análisis y manipulación de datos, por lo que es muy útil para crear tablas de resultados.

4.2.3. Numpy

Como se manipularon en este trabajo imágenes que se interpretan como matrices, se hizo el uso de Numpy ya que este se enfoca en matrices y vectores.

4.2.4. Conversión de una imagen RGB a luminancia

Primero antes de obtener los índices de una imagen es necesario procesarla, para ello se convierte la imagen RGB en una sola matriz que nos representa la luminancia de cada píxel, para así aplicar los métodos siguientes [8].

Se tomaran de la imagen los valores de un píxel en la matriz Red, Green y Blue para después aplicar la formula 2.19. Donde como resultado tendremos una matriz de el mismo tamaño de la imagen de entrada.

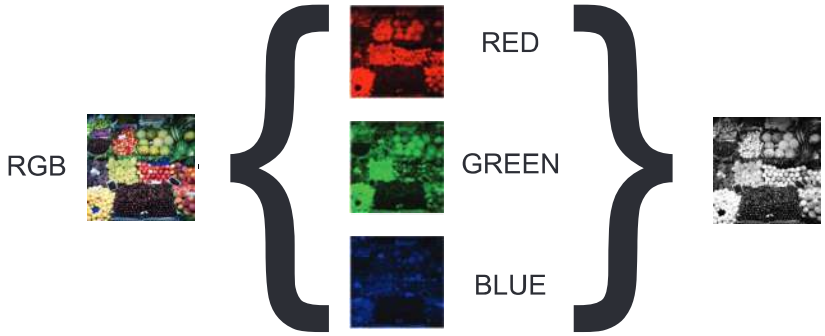

Figura 4.3: Imagen RGB a escala de grises

Como se observa en la figura 4.3, tenemos una imagen RGB con sus 3 matices y esta es transformada a escala de grises, donde solo tiene una matriz donde en donde el valor de cada píxel equivale a una graduación de gris, que con esto, indica también que tan luminoso es un píxel donde blanco significa luminosidad alta y negro es luminosidad baja.

4.3. Algoritmo para obtención del índice de una imagen

En la utilización de estos algoritmos todos toman a la imagen transformada en luminancia.

4.3.1. Luminancia de una imagen

Este índice representa la luminosidad de toda la imagen.
Para el calculo de un índice que nos diga que tan luminosa es una imagen, realizamos la sumatoria de todos los píxeles para después dividirlo entre la cantidad total de píxeles y además dividirlo por 255, ya que una imagen totalmente blanca nos dará un índice de 255, esto para normalizar el índice y quede en un rango entre 0 y 1.

Algorithm 1 Calculo de Luminancia en una foto

```
luminancia ← 0
for i ← 0 to ancho_imagen do
    for i ← 0 to ancho_imagen do
        luminancia ← luminancia + imagen[i][j]
    end for
end for
luminancia ← luminancia/(ancho_imagen ∗ alto_imagen ∗ 255)
return luminancia
```

Aplicando este algoritmo en python a imágenes podemos ver una comparativa de la imagen mas luminosa según el algoritmo y la menor, como se muestra en la imagen 4.5.

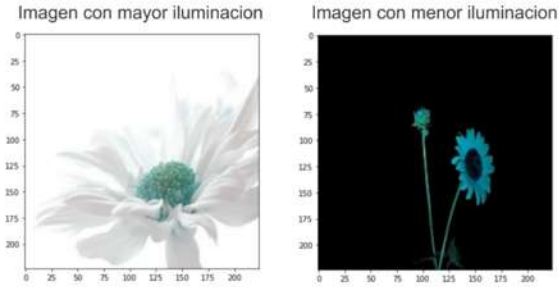

Figura 4.4: Comparación del índice de iluminación

4.3.2. Desviación Estándar

Este índice representa que tan dispersos están los valores de los píxeles de toda la imagen de la media de estos valores. Esto indica también el nivel de ruido ambiental que tiene una imagen [8].

Para la realización de este algoritmo se toma la imagen en luminancia para después sacar la media aritmética de todos sus valores, como viene dado por la formula siguiente.

$$\sigma = \sqrt{\frac{\Sigma_{i=1}^{n}(x_i - \bar{a})}{n}} \tag{4.1}$$

Aplicando la desviación estándar tomando cada uno de los píxeles de la imagen, obtenemos una comparación de una imagen con alta desviación estándar y con una de baja desviación estándar.

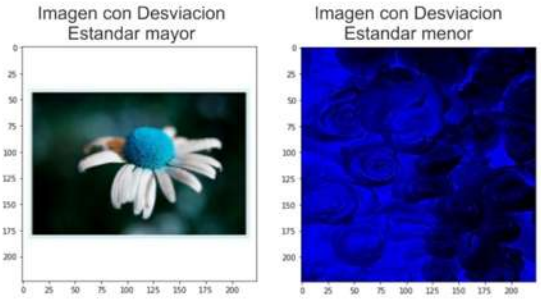

Figura 4.5: Comparación de desviación estándar

4.3.3. Variabilidad del color

Se propuso este algoritmo para obtener la variabilidad de el color en una imagen, partiendo desde que una imagen ruidosa debe contener cambios grandes de color en una área pequeña de píxeles.

Se partió de la operación de convolución, ya que toma una área de píxeles. Partiendo desde el píxel central se compara que tanto varia su color con los píxeles vecinos, para esto se hace la diferencia entre el píxel central y el promedio de los vecinos dentro de el área especificada como se muestra en la figura siguiente.

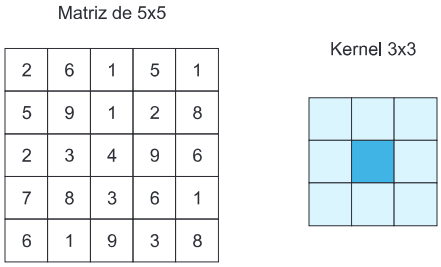

Figura 4.6: Vecinos de un píxel con kernel 3x3

En la imagen 4.6, dentro de el kernel vemos que el valor central es de color mas fuerte, los de color mas débil son los vecinos de el píxel central utilizando una ventana de 3x3.

Esta área a tomar de píxeles por simplicidad y su parecido con la operación de convolución se llamo kernel, tal como lo veíamos en capítulos anteriores, y su tamaño definirá cuantos vecinos tomara para la comparación de un píxel.

La comparación de el píxel con sus vecinos viene dado por tamaño del kernel.

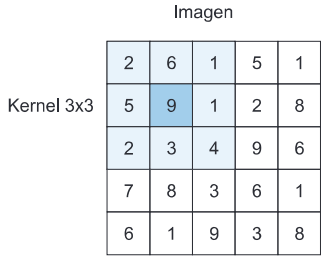

Figura 4.7: Vecinos de un píxel con kernel 3x3

Como se observa en la figura 4.7 el píxel que se toma como pivote es el de el centro, el que tiene el color mas fuerte, y los vecinos son los de color mas bajo, es decir el central es

el numero 9 y los vecinos de el 9 serian los números 2, 6, 1, 5, 1, 2, 3, 4.

Para este calculo se hará la diferencia entre el pivote y la media aritmética de los vecinos como se muestra enseguida.

$$\begin{aligned} Variabilidad &= |Valor\_central - promedio\_vecinos| \\ &= |9 - \frac{(2+6+1+5+1+2+3+4)}{8}| \\ &= |9 - \frac{(24)}{8}| \\ &= |9-3| = 6 \end{aligned} \tag{4.2}$$

Este numero nos indicara la variabilidad, un numero alto significa que el pivote tiene valor mas distante a la media de sus vecinos, y un valor cercano a 0 significa que el pivote y sus vecinos tienen valores muy cercanos.

Esto ha sido aplicado hasta ahora en un solo píxel, lo que se desea saber es la variabilidad de toda la imagen, para esto habrá que recorrer el kernel con un paso = 1 hasta recorrer toda la imagen, esto como podemos ver es un proceso iterativo, y el promedio de esta sumatoria indica el índice de la imagen completa, como se puede ver en la imagen siguiente.

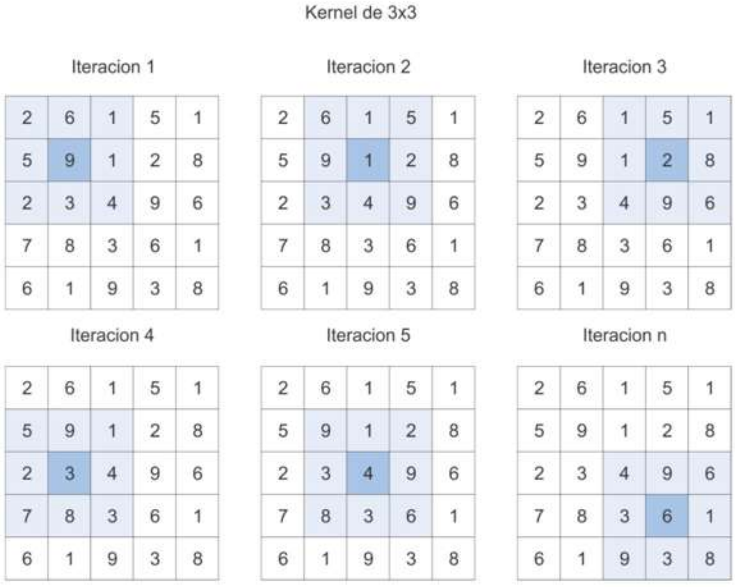

Figura 4.8: Índice variabilidad de la imagen completa

Este algoritmo aplicado nos dice que tantos cambios de colores en los píxeles en una área pequeña tiene, para verlo de una manera mas gráfica, veremos 2 imágenes a las que se les calculo su índice de variabilidad las que podremos observar en la imagen siguiente.

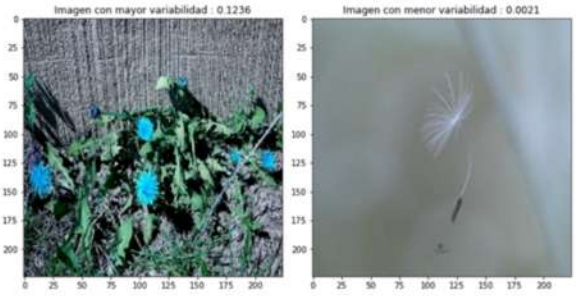

Figura 4.9: Comparación índice variabilidad en imágenes

Variando el tamaño del kernel

El algoritmo propuesto recibe como argumento el tamaño de el kernel, este debe ser un numero impar para poder realizar la operación (3, 5, 7, 9, ...), un tamaño mas grande en el kernel significa que tomara una área mas grande de vecinos con los cuales se va a comparar, se observa en la figura un kernel de 5x5, y así este puede ser de el tamaño deseado.

Kernel 5x5

2	6	1	5	1	3	8
4	9	3	7	9	5	2
9	3	4	8	7	4	7
6	4	7	9	2	9	5
3	2	8	1	6	7	3
7	5	1	9	8	1	4
8	1	2	7	3	6	6

Figura 4.10: Variabilidad de color con kernel 5x5

Se muestra en la figura 4.10 un kernel de 5x5, donde el área de color mas bajo serian todos los vecinos de ese píxel (el color mas fuerte) tomando una ventana de 5x5.

4.4. Separación de el conjunto de imágenes

En este trabajo de tesis se usara el 80 % de las imágenes para el entrenamiento y el 20 % restante se utilizara para la evaluación. Únicamente a las imágenes de evaluación se les calculo los índices ya vistos en la sección anterior (luminosidad, desviación estándar, variabilidad de color).

Una vez obtenido el conjunto de imágenes con sus respectivos índices para cada imagen, se crearon las listas que contienen ordenadas las imágenes de acuerdo a:

- Índice de luminosidad
- Índice de Desviación estándar
- Índice de Variabilidad del color

Con las listas creadas ahora se dividirá cada lista en 2, imágenes de alto índice y de bajo índice, quedando como resultado 6 listas que son las siguientes:

- Bajo índice de Luminosidad
- Alto índice de Luminosidad
- Bajo índice de Desviación
- Alto índice de Desviación
- Bajo índice de Variabilidad de color
- Alto índice de Variabilidad de color

Cada una de estas listas se utilizó para la evaluación de cada uno de los modelos en clasificación.

4.5. Servicio de computo Kaggle

Se utilizo los servicios de computo que brinda la pagina de Kaggle, se creo una cuenta y aquí mismo es donde se encuentran todos los códigos utilizados en el proyecto.

4.6. Implementación de Redes Neuronales Convolucionales Pre entrenadas para el conjunto de flores

Como se vio en el capitulo 4, la estructura de cada una de las redes preentrenadas utilizadas, se utilizo cada una de ellas para la clasificación de el conjunto de imágenes de flores.

Además de la estructura propia de cada red pre-entrenada se agrego una capa de flatten, y una capa densa para la clasificación, en la figura siguiente se muestra la estructura de cada una de las redes preentrenadas que se utilizaron como se puede observar en la imagen.

Modelo Secuencial
Modelo base
Flatten
Dense (5)

Cuadro 4.1: Estructura modelos para entrenamiento con flores

En la tabla 4.1, vemos la estructura general que se utilizara para el conjunto de flores, donde:

- Modelo Base: Este es el modelo pre-entrenado (Vgg19, Resnet50, Inception, Xception, DenseNet, EfficientNet)
- Flatten: Esta es la capa de aplanado, donde una matriz se convierte en un array de una dimensión
- Dense(5): En esta capa se utilizo 5 unidades Densas ya se tienen 5 clases en el conjunto de flores, y además tiene una función de activación Softmax, para así hacer la clasificación de las imágenes.

Ya creado el modelo y definida su estructura ahora se procederá a definir los parámetros para el compilado de el modelo.

Se utilizo el optimizador Adam, por su adaptabilidad a cualquier problema y por ser el mas utilizado en la actualidad además de la función de perdida de Perdida de entropía cruzada (categorical cross entropy), ya que esta es ideal para la clasificación múltiple como vimos en el capitulo 2.2.4, y en este caso se tiene 5 clases.

Para el entrenamiento se utilizo un tamaño de lote de 64, y este llevara acabo 12 épocas. Se utilizo el subconjunto del 80 % de las imágenes totales para el entrenamiento, donde los únicos parámetros entrenables que se dejaron fueron los de la capa densa de al final, dejando con los pesos de imagenet todas las redes preentrenadas.

4.6.1. Evaluacion

A los modelos preentrenados se les pasaron los subconjuntos de imágenes que se ha visto anteriormente. Se le paso una de estas listas de imágenes y se propuso que el modelo intentara predecir de que clase era la imagen de entrada, y si el modelo predice de manera correcta la clase de la imagen de entrada aumenta la precisión, para al final quedarnos con un porcentaje de que tantas acertó en la predicción.

4.7. Implementación de Redes Neuronales Convolucionales Pre entrenadas para imagenes de ImageNet

Ahora se tomaron las redes preentrenadas evaluándolas con las imagenes que fueron probadas su eficiencia.

Como se comento en capítulos anteriores imagenet, son miles de imágenes las que se utilizaron para la evaluación de estas redes, como son tantas y tomaría demasiado tiempo experimentar con todas,por cuestiones de tiempo se tomaron únicamente 6000 imágenes, con las que se evaluaran.

Primero a estas 6000 imágenes se les calculo el índice de variabilidad de color, desviación estándar y luminosidad como se vio la sección 4.3.

Como se muestra a continuación la gráfica 4.11 se muestra muestra la distribución que tienen las imágenes con los diferentes índices calculados.

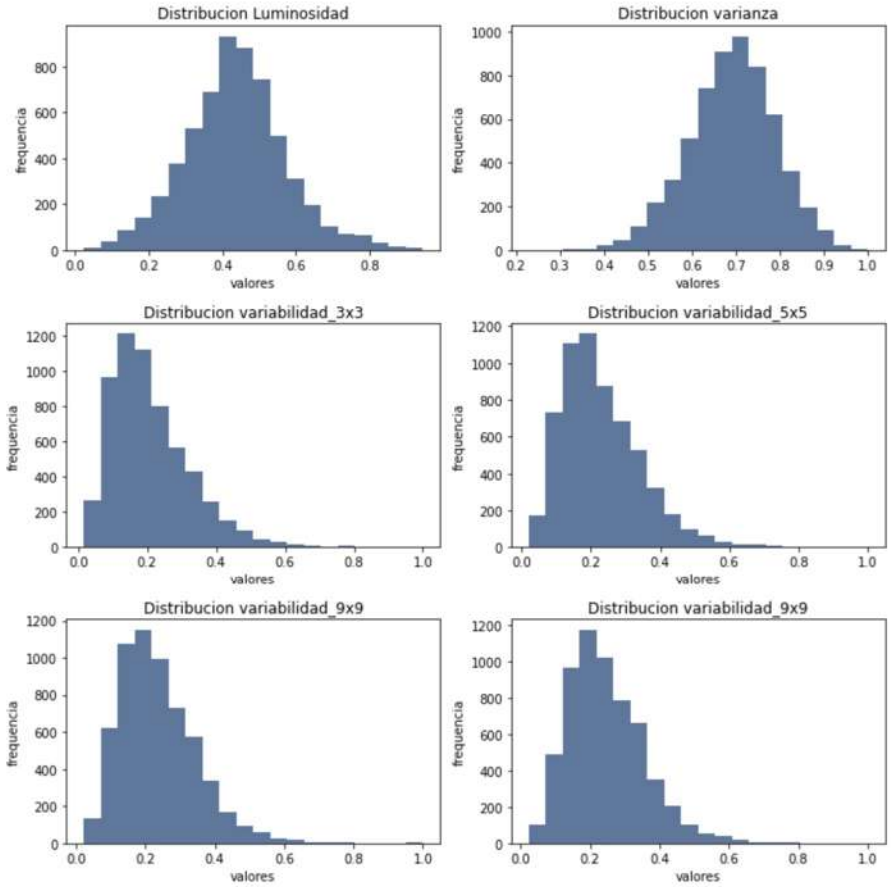

Figura 4.11: Distribución de las imágenes en imagenet

Se puede notar que los índices van en un rango de [0,1], después se dividió el conjunto para tomar las 2000 imágenes de índice bajo y las 2000 imágenes de índice alto, eliminando las 2000 imágenes de el medio, como se muestra en la figura 4.12.

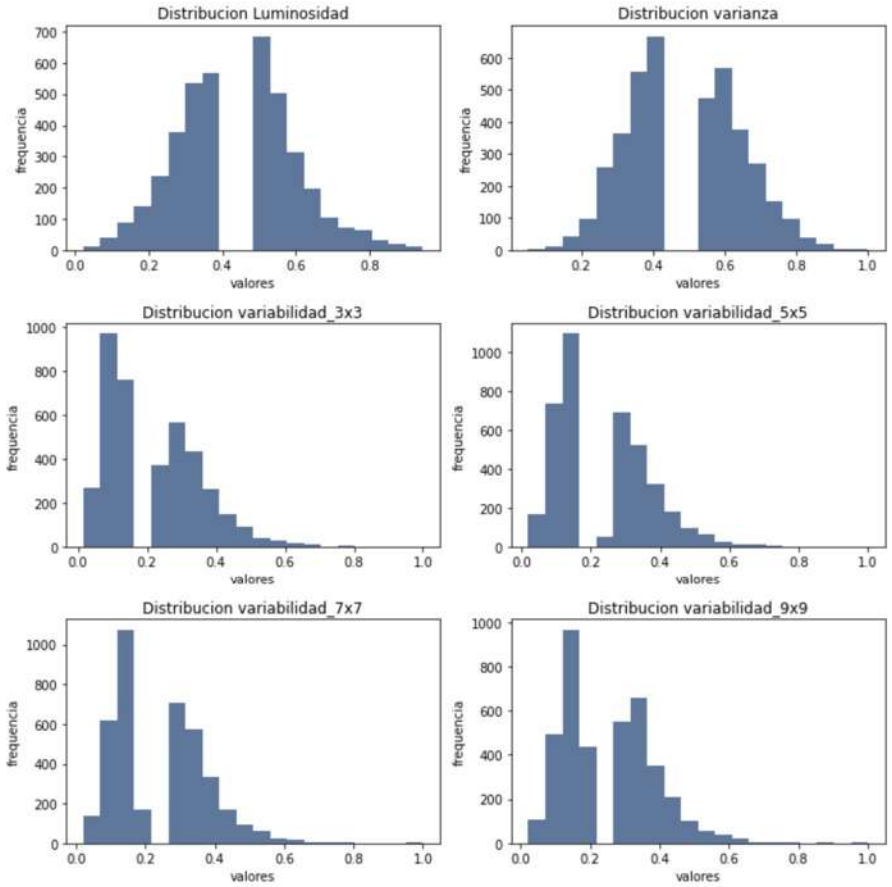

Figura 4.12: Distribución de las imágenes en imagenet alto y bajo índice

Después se realizo la evaluación de las redes neuronales convolucionales preentrenadas, con cada uno de los subconjuntos creados, tomando en cuenta la cantidad de veces que lograron acertar a la imagen de prueba.

5 Pruebas y Resultados

En este capítulo se presentan las pruebas que se realizaron sobre las redes neuronales diseñadas. Con el objetivo de obtener la precisión que tienen los modelos al predecir imágenes con distintos índices.

Se realizaron las pruebas para ambos modelos propuestos en el capitulo anterior los cuales son:

- Clasificación de imágenes de flores
- Clasificación de imágenes de ImageNet

Los cuales serán expuestos a continuación.

5.1. Clasificación de flores

Se hizo una comparación de cada una de listas respecto a las otras, para ver si variaban las imágenes de una lista a otra y se obtuvo la siguiente tabla.

lista	varianza_list	luminosidad_list	variabilidad_3x3_list	variabilidad_5x5_list	variabilidad_7x7_list	variabilidad_9x9_list
varianza_list	0	181	190	186	185	182
luminosidad_list	181	0	210	208	208	208
variabilidad_3x3_list	190	210	0	27	41	49
variabilidad_5x5_list	186	208	27	0	17	26
variabilidad_7x7_list	185	208	41	17	0	10
variabilidad_9x9_list	182	208	49	26	10	0

Cuadro 5.1: Diferencias entre listas

Con esta tabla podemos afirmar que no son las mismas imágenes en todas las listas, que varían en una cierta cantidad de imágenes respecto a las demás.

Como se explico en la sección 4.6, se utilizo la estructura de la tabla 4.1 para los modelos a evaluar.

Se evaluaron todos los modelos Vgg19, Resnet50, Inception, Xception, DenseNet, EfficientNet con todos los conjuntos de imágenes descritos y se mostraran los resultados a continuación por índice.

5.1.1. Resultados con Luminancia

Se evaluaron los modelos con las listas de alto y bajo índice de luminancia en la clasificación y se obtuvieron los siguientes resultados que se muestran en la tabla 5.2.

modelo	perdida	precisión	modo
DenseNet	**16.170729**	**0.133949**	**Luminosidad_mayor**
Vgg19	3.800770	0.150115	Luminosidad_mayor
Xception	17.356699	0.154734	Luminosidad_mayor
Inception	12.616340	0.154734	Luminosidad_mayor
Resnet	2.262680	0.166282	Luminosidad_mayor
EfficientNet	1.659910	0.168591	Luminosidad_mayor
EfficientNet	1.652488	0.187500	Luminosidad_menor
Xception	15.187394	0.252315	Luminosidad_menor
DenseNet	14.362477	0.268519	Luminosidad_menor
Inception	10.583950	0.268519	Luminosidad_menor
Resnet	2.072828	0.275463	Luminosidad_menor
Vgg19	**3.365324**	**0.282407**	**Luminosidad_menor**

Cuadro 5.2: Rendimiento de acuerdo con la iluminación

Como se observa en la tabla 5.2 se ordenaron de acuerdo a la precisión de menor a mayor, siendo la ultima fila el registro con mayor precisión.

Se muestra que el modelo Vgg19 utilizando la lista de luminosidad menor obtuvo la mejor precisión.

A su vez el modelo DenseNet obtuvo la menor precisión utilizando la lista de luminosidad mayor.

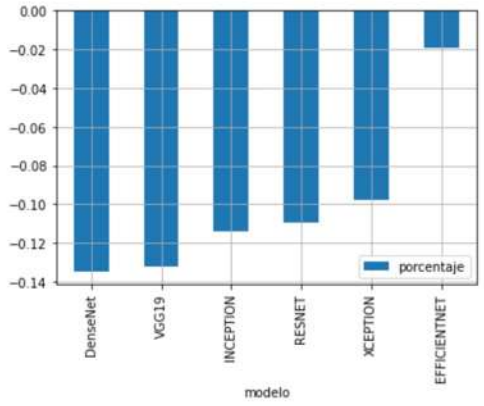

Figura 5.1: Gráfico de barras de acuerdo con la luminosidad

La figura 5.1 se obtuvo, realizando la diferencia de las precisiones, con el índice alto menos el índice menor, por lo que si tenemos una barra negativa indica que se comporto mejor con las de índice bajo, y si es una gráfica positiva significa que ha tenido un mayor rendimiento con imágenes de alto índice.
De acuerdo con la figura 5.1, podemos observar que se obtuvo una mejor precisión cuando eran evaluados con imágenes de menor luminosidad y todos los modelos siguen este comportamiento.

5.1.2. Resultados con Varianza

Al evaluar los modelos con las listas de alto y bajo indice de varianza se obtuvo la tabla siguiente.

modelo	perdida	precisión	modo
EfficientNet	**1.705594**	**0.096998**	**Varianza_mayor**
Xception	17.384386	0.175926	Varianza_menor
DenseNet	16.437334	0.180556	Varianza_menor
Vgg19	4.022330	0.194444	Varianza_menor
Inception	12.277884	0.194444	Varianza_menor
Resnet	2.438555	0.212963	Varianza_menor
DenseNet	14.100665	0.221709	Varianza_mayor
Resnet	1.897797	0.228637	Varianza_mayor
Inception	10.926320	0.228637	Varianza_mayor
Xception	15.164782	0.230947	Varianza_mayor
Vgg19	3.145281	0.237875	Varianza_mayor
EfficientNet	**1.606699**	**0.259259**	**Varianza_menor**

Cuadro 5.3: Rendimiento de acuerdo con la varianza

Se muestra en la tabla 5.3, el modelo EfficientNet obtuvo la mejor precisión utilizando la lista de Varianza menor, y a su vez obtuvo la menor precisión utilizando la lista de Varianza mayor.

Se analizaron los resultados y se muestra que todos los modelos restantes tuvieron un comportamiento diferente, ya que con imágenes de Varianza menor obtuvieron una menor eficiencia, y con la lista de varianza mayor, obtuvieron un rendimiento mayor.

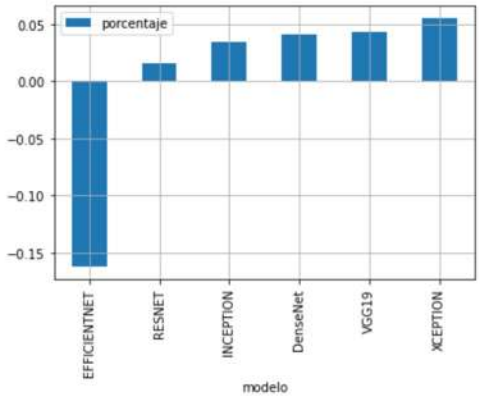

Figura 5.2: Gráfico de barras de acuerdo con la luminosidad

Analizando los resultados que se muestran en la figura 5.2 que todos los modelos obtuvieron una mejor precisión cuando las imágenes tenían una varianza mayor, esto a

su vez indica que se comporta mejor con imágenes que no son mayormente de el mismo color, si no que es mejor cuando las imágenes tienen colores mas variados.

5.1.3. Resultados con Variabilidad de color

Ahora se puso a prueba el algoritmo propuesto para la variabilidad de el color, este se puso a prueba con un kernel de tamaño 3x3, 5x5, 7x7, 9x9, con lo cual se obtienen las siguientes tablas.

Variabilidad con kernel 3x3

modelo	perdida	precision	modo
DenseNet	**16.176565**	**0.131640**	**Variabilidad3x3_mayor**
XCEPTION	18.087147	0.131640	Variabilidad3x3_mayor
VGG19	3.688847	0.140878	Variabilidad3x3_mayor
EFFICIENTNET	1.683077	0.143519	Variabilidad3x3_menor
INCEPTION	12.924432	0.147806	Variabilidad3x3_mayor
RESNET	2.035732	0.189376	Variabilidad3x3_mayor
EFFICIENTNET	1.629392	0.212471	Variabilidad3x3_mayor
RESNET	2.300300	0.252315	Variabilidad3x3_menor
DenseNet	14.356628	0.270833	Variabilidad3x3_menor
XCEPTION	14.455257	0.275463	Variabilidad3x3_menor
INCEPTION	10.275145	0.275463	Variabilidad3x3_menor
VGG19	**3.477507**	**0.291667**	**Variabilidad3x3_menor**

Cuadro 5.4: Rendimiento de acuerdo con la Variabilidad con filtro 3x3

Al observar la tabla 5.4, el modelo Vgg19 obtuvo la mejor precisión utilizando la lista de variabilidad menor, y el modelo de DenseNet obtuvo la menor precisión utilizando la lista de variabilidad mayor.

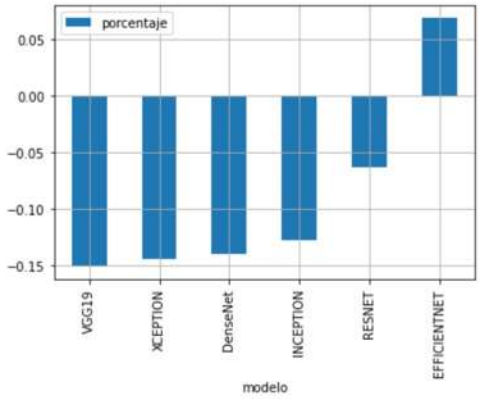

Figura 5.3: Gráfico de barras de acuerdo con la variabilidad con tamaño 3x3

Como se observa en la figura 5.3, la mayoría de los modelos se comportaron mejor cuando se utilizaron imágenes con bajo índice de variabilidad de color.

Variabilidad con kernel 5x5

modelo	perdida	presicion	modo
DenseNet	**16.821602**	**0.136259**	**Variabilidad5x5_mayor**
XCEPTION	18.766663	0.140878	Variabilidad5x5_mayor
VGG19	3.715980	0.150115	Variabilidad5x5_mayor
INCEPTION	13.059269	0.154734	Variabilidad5x5_mayor
EFFICIENTNET	1.670771	0.157407	Variabilidad5x5_menor
RESNET	2.018030	0.193995	Variabilidad5x5_mayor
EFFICIENTNET	1.641670	0.198614	Variabilidad5x5_mayor
RESNET	2.318044	0.247685	Variabilidad5x5_menor
DenseNet	13.710097	0.266204	Variabilidad5x5_menor
XCEPTION	13.774168	0.266204	Variabilidad5x5_menor
INCEPTION	10.139997	0.268519	Variabilidad5x5_menor
VGG19	**3.450311**	**0.282407**	**Variabilidad5x5_menor**

Cuadro 5.5: Rendimiento de acuerdo con la Variabilidad con filtro 5x5

Con el tamaño de kernel de 5x5 se tiene que el modelo con mayor precisión es el Vgg19 con la lista de Variabilidad menor, y el modelo que obtuvo menor precisión fue el DenseNet utilizando la lista de variabilidad mayor.

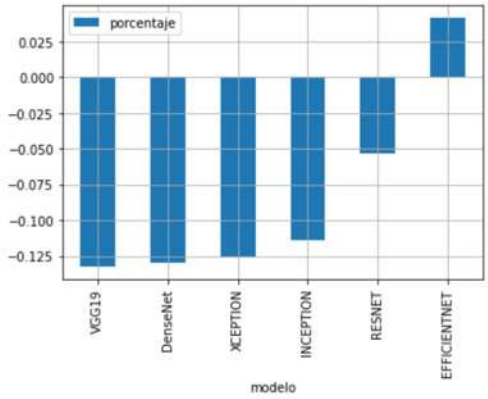

Figura 5.4: Gráfico de barras de acuerdo con la variabilidad con tamaño 5x5

Como se observa en la figura 5.4, la mayoría de los modelos se comportaron mejor cuando se utilizaron imágenes con bajo índice de variabilidad de color.

Variabilidad con kernel 7x7

modelo	perdida	presicion	modo
DenseNet	**16.484173**	**0.157044**	**Variabilidad7x7_mayor**
XCEPTION	18.496828	0.157044	Variabilidad7x7_mayor
VGG19	3.610389	0.168591	Variabilidad7x7_mayor
INCEPTION	12.872491	0.170901	Variabilidad7x7_mayor
EFFICIENTNET	1.659452	0.173611	Variabilidad7x7_menor
EFFICIENTNET	1.652963	0.182448	Variabilidad7x7_mayor
RESNET	1.968380	0.200924	Variabilidad7x7_mayor
RESNET	2.367808	0.240741	Variabilidad7x7_menor
DenseNet	14.048309	0.245370	Variabilidad7x7_menor
XCEPTION	14.044629	0.250000	Variabilidad7x7_menor
INCEPTION	10.327205	0.252315	Variabilidad7x7_menor
VGG19	**3.556146**	**0.263889**	**Variabilidad7x7_menor**

Cuadro 5.6: Rendimiento de acuerdo con la Variabilidad con filtro 7x7

Con el tamaño de kernel de 7x7 se tiene que el modelo con mayor precisión es el Vgg19 con la lista de Variabilidad menor, y el modelo que obtuvo menor precisión fue el DenseNet utilizando la lista de variabilidad mayor.

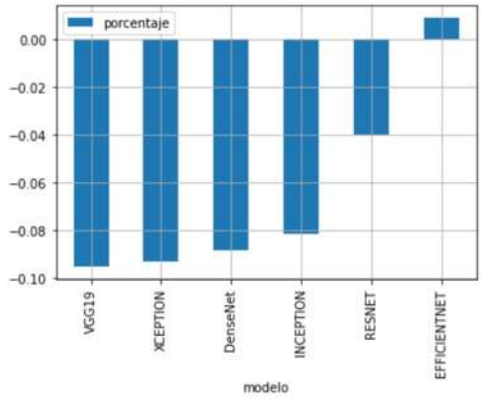

Figura 5.5: Gráfico de barras de acuerdo con la variabilidad con tamaño 7x7

Como se observa en la figura 5.5, la mayoría de los modelos se comportaron mejor cuando se utilizaron imágenes con bajo índice de variabilidad de color.

Variabilidad con kernel 9x9

modelo	perdida	presicion	modo
DenseNet	**16.552851**	**0.159353**	**Variabilidad9x9_mayor**
XCEPTION	18.350309	0.159353	Variabilidad9x9_mayor
VGG19	3.586711	0.166282	Variabilidad9x9_mayor
EFFICIENTNET	1.658116	0.173210	Variabilidad9x9_mayor
INCEPTION	12.820805	0.173210	Variabilidad9x9_mayor
EFFICIENTNET	1.654287	0.182870	Variabilidad9x9_menor
RESNET	1.949026	0.200924	Variabilidad9x9_mayor
RESNET	2.387207	0.240741	Variabilidad9x9_menor
DenseNet	13.979469	0.243056	Variabilidad9x9_menor
XCEPTION	14.191487	0.247685	Variabilidad9x9_menor
INCEPTION	10.379014	0.250000	Variabilidad9x9_menor
VGG19	**3.579879**	**0.266204**	**Variabilidad9x9_menor**

Cuadro 5.7: Rendimiento de acuerdo con la Variabilidad con filtro 9x9

Con un tamaño de kernel de 9x9 se tiene que el modelo con mayor precisión es el Vgg19 con la lista de variabilidad menor, y el modelo que obtuvo menor precisión fue el DenseNet utilizando la lista de variabilidad mayor.

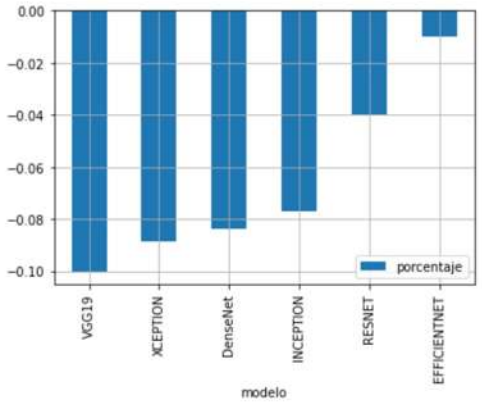

Figura 5.6: Gráfico de barras de acuerdo con la variabilidad con tamaño 9x9

Como se observa en las tablas obtenidas, los modelos siguen un comportamiento y es que tienen un mayor rendimiento cuando las imágenes tienen una variabilidad de color baja, lo que indica que tienen cambios suaves entre píxeles.

Además de esto vemos que al aumentar el tamaño de el kernel se mantienen en el mismo orden los modelos pero reduce poco a poco la precisión.

5.2. Clasificación de imágenes de ImageNet

Se realizo la evaluación de las redes neuronales con las imágenes de imageNet, con lo cual se obtuvieron las siguientes tablas

Se evaluaron los modelos Vgg19, Resnet50, Inception, Xception, DenseNet, EfficientNet, con los conjuntos de imágenes explicados anteriormente en el capitulo 4.7

5.2.1. Resultados con el conjunto completo

Primero se evaluaron los modelos con el conjunto de imágenes completo para poder conocer el porcentaje de aciertos que se tiene como se observa en la tabla 5.8.

modelo	aciertos	porcentaje	modo
Vgg19	**2294**	**0.57350**	**Evaluacion completa**
Inception	2369	0.59225	Evaluacion completa
Exception	2412	0.60300	Evaluacion completa
DenseNet	2425	0.60625	Evaluacion completa
Resnet	2469	0.61725	Evaluacion completa
EfficientNet	**2496**	**0.62400**	**Evaluacion completa**

Cuadro 5.8: Rendimiento de acuerdo con la Varianza

Se observa que el modelo con mayor porcentaje de aciertos fue el modelo EfficientNet , y el que obtuvo el menor porcentaje fue el modelo Vgg19.

5.2.2. Resultados con Varianza

Se evaluaron los modelos con las listas de alto y bajo indice en la clasificacion, se obtuvieron los resultados que se muestran en la tabla.

modelo	aciertos	porcentaje	modo
Vgg19	**1126**	**0.5630**	**varianza_mayor**
Vgg19	1193	0.5965	varianza_menor
Resnet	1199	0.5995	varianza_mayor
Inception	1200	0.6000	varianza_mayor
Xception	1209	0.6045	varianza_mayor
Inception	1215	0.6075	varianza_menor
DenseNet	1234	0.6170	varianza_mayor
DenseNet	1235	0.6175	varianza_menor
Xception	1236	0.6180	varianza_menor
EfficientNet	1245	0.6225	varianza_mayor
EfficientNet	1278	0.6390	varianza_menor
Resnet	**1304**	**0.6520**	**varianza_menor**

Cuadro 5.9: Rendimiento de acuerdo con la Varianza

Para cada evaluación se utilizo un subconjunto de 2000 imágenes, y la cantidad de aciertos es la cantidad de imágenes que clasifico correctamente.

Como se observa en la tabla 5.9, los resultados están ordenados por la cantidad de aciertos, se ve que el modelo que menor porcentaje obtuvo fue el Vgg19 con las imágenes de varianza mayor, y el que tuvo un mayor porcentaje fue el Resnet con las imágenes de

varianza menor.

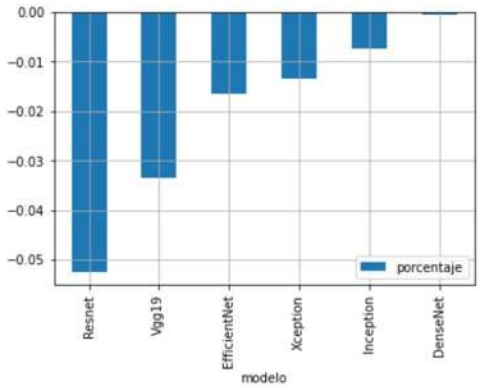

Figura 5.7: Gráfico de barras de acuerdo con la varianza

Se observa mas claramente en la figura 5.7 que para cada modelo tienen una mayor precisión cuando utilizan las imágenes de varianza menor.

5.2.3. Resultados con Iluminación

modelo	aciertos	porcentaje	modo
Vgg19	**1112**	**0.5560**	**luminosidad_menor**
Inception	1167	0.5835	luminosidad_menor
DenseNet	1177	0.5885	luminosidad_menor
Vgg19	**1182**	**0.5910**	**luminosidad_mayor**
Xception	1191	0.5955	luminosidad_menor
Inception	**1202**	**0.6010**	**luminosidad_mayor**
EfficientNet	**1203**	**0.6015**	**luminosidad_menor**
Xception	**1221**	**0.6105**	**luminosidad_mayor**
Resnet	**1228**	**0.6140**	**luminosidad_menor**
Resnet	**1241**	**0.6205**	**luminosidad_mayor**
DenseNet	**1248**	**0.6240**	**luminosidad_mayor**
EfficientNet	**1293**	**0.6465**	**luminosidad_mayor**

Cuadro 5.10: Rendimiento de acuerdo con la luminosidad

Como se observa en la tabla 5.10, el modelo con un porcentaje mayor fue EfficientNet con las imágenes de luminosidad mayor, y el peor fue el modelo Vgg19 con el modo de luminosidad menor.

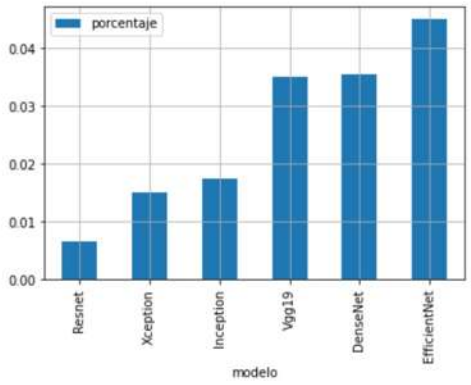

Figura 5.8: Gráfico de barras de acuerdo con la luminancia

Como se muestra en la figura 5.8, todos los modelos tienen un mayor rendimiento cuando se evaluaron con las imágenes de luminosidad mayor.

5.2.4. Resultados con Variabilidad de color

Se puso a prueba el algoritmo propuesto con la variabilidad de color utilizando un kernel de tamaño 3x3, 5x5, 7x7, 9x9, con lo que se obtuvieron las siguientes tablas.

Variabilidad con kernel 3x3

modelo	aciertos	porcentaje	modo
Vgg19	**1156**	**0.5780**	**variabilidad3x3_menor**
Inception	1165	0.5825	variabilidad3x3_mayor
Vgg19	1189	0.5945	variabilidad3x3_mayor
DenseNet	1192	0.5960	variabilidad3x3_mayor
Xception	1196	0.5980	variabilidad3x3_mayor
Resnet	1205	0.6025	variabilidad3x3_mayor
Inception	1225	0.6125	variabilidad3x3_menor
EfficientNet	1228	0.6140	variabilidad3x3_mayor
Xception	1247	0.6235	variabilidad3x3_menor
DenseNet	1260	0.6300	variabilidad3x3_menor
Resnet	1279	0.6395	variabilidad3x3_menor
EfficientNet	**1287**	**0.6435**	**variabilidad3x3_menor**

Cuadro 5.11: Rendimiento de acuerdo con la Variabilidad con filtro 3x3

Se observa en la tabla 5.11, que el modelo con mayor porcentaje fue EficientNet con el conjunto de imágenes con variabilidad menor, y el modelo con menor fue Vgg19 con el conjunto de imágenes de variabilidad menor.

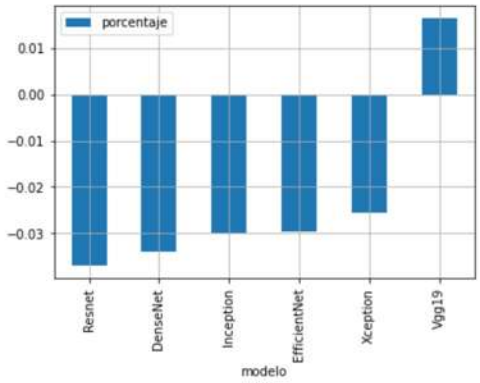

Figura 5.9: Gráfico de barras de acuerdo con la variabilidad de tamaño 3x3

Además en el gráfico 5.9, se observa que se obtuvo un mayor rendimiento en los modelos cuando se tenían imágenes con un bajo índice de variabilidad.

Variabilidad con kernel 5x5

modelo	aciertos	porcentaje	modo
Vgg19	**1150**	**0.5750**	**variabilidad5x5_menor**
Inception	1154	0.5770	variabilidad5x5_mayor
Xception	1177	0.5885	variabilidad5x5_mayor
DenseNet	1181	0.5905	variabilidad5x5_mayor
Vgg19	1181	0.5905	variabilidad5x5_mayor
Resnet	1200	0.6000	variabilidad5x5_mayor
Inception	**1207**	**0.6035**	**variabilidad5x5_menor**
EfficientNet	**1214**	**0.6070**	**variabilidad5x5_mayor**
Xception	**1240**	**0.6200**	**variabilidad5x5_menor**
DenseNet	**1264**	**0.6320**	**variabilidad5x5_menor**
Resnet	**1270**	**0.6350**	**variabilidad5x5_menor**
EfficientNet	**1281**	**0.6405**	**variabilidad5x5_menor**

Cuadro 5.12: Rendimiento de acuerdo con la Variabilidad con filtro 5x5

Se observa en la tabla 5.12 que el modelo con mayor porcentaje de aciertos fue EfficientNet utilizando las imágenes de variabilidad menor, así como también el que obtuvo el menor porcentaje fue el modelo Vgg19 con las imágenes de variabilidad menor.

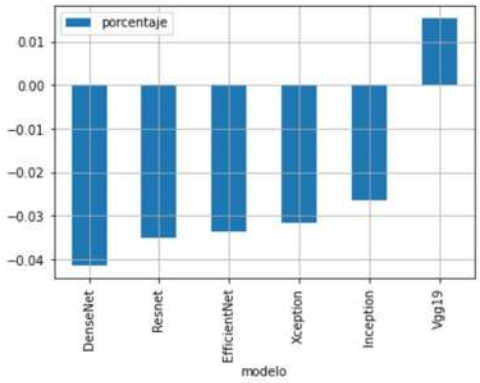

Figura 5.10: Gráfico de barras de acuerdo con la variabilidad de tamaño 5x5

Como se ve en el gráfico 5.10, se observa que se obtuvo un mayor rendimiento en los modelos cuando se tenían imágenes con un bajo índice de variabilidad.

Variabilidad con kernel 7x7

modelo	aciertos	porcentaje	modo
Inception	**1149**	**0.5745**	**variabilidad7x7_mayor**
Vgg19	1155	0.5775	variabilidad7x7_menor
Vgg19	1172	0.5860	variabilidad7x7_mayor
Xception	1173	0.5865	variabilidad7x7_mayor
DenseNet	1179	0.5895	variabilidad7x7_mayor
Resnet	1194	0.5970	variabilidad7x7_mayor
Inception	1203	0.6015	variabilidad7x7_menor
EfficientNet	1206	0.6030	variabilidad7x7 mayor
Xception	1242	0.6210	variabilidad7x7 menor
DenseNet	1256	0.6280	variabilidad7x7_menor
Resnet	1273	0.6365	variabilidad7x7_menor
EfficientNet	**1284**	**0.6420**	**variabilidad7x7_menor**

Cuadro 5.13: Rendimiento de acuerdo con la Variabilidad con filtro 7x7

Se muestra en la tabla 5.13 el modelo que obtuvo mayor porcentaje de aciertos fue EfficientNet utilizando las imágenes de variabilidad menor, así mismo el que obtuvo el

menor porcentaje de imágenes fue Inception utilizando las imágenes con variabilidad mayor.

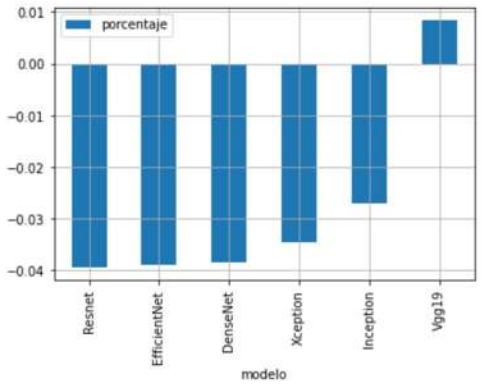

Figura 5.11: Gráfico de barras de acuerdo con la variabilidad de tamaño 7x7

En el gráfico 5.11, se observa que se obtuvo un mayor rendimiento en los modelos cuando se tenían imágenes con un bajo índice de variabilidad.

Variabilidad con kernel 9x9

modelo	aciertos	porcentaje	modo
Vgg19	**1150**	**0.5750**	**variabilidad9x9_menor**
Inception	1157	0.5785	variabilidad9x9_mayor
Vgg19	1173	0.5865	variabilidad9x9_mayor
Xception	1177	0.5885	variabilidad9x9_mayor
DenseNet	1184	0.5920	variabilidad9x9_mayor
Inception	**1198**	**0.5990**	**variabilidad9x9_menor**
Resnet	1202	0.6010	variabilidad9x9_mayor
EfficientNet	1206	0.6030	variabilidad9x9_mayor
Xception	**1242**	0.6210	**variabilidad9x9_menor**
DenseNet	1256	0.6280	**variabilidad9x9_menor**
Resnet	1265	0.6325	**variabilidad9x9_menor**
EfficientNet	**1279**	**0.6395**	**variabilidad9x9_menor**

Cuadro 5.14: Rendimiento de acuerdo con la Variabilidad con filtro 9x9

Se ve en la tabla 5.14 que el modelo con mayor porcentaje de aciertos fue EfficientNet utilizando las imágenes de variabilidad menor y el modelo que obtuvo el menor porcentaje de aciertos fue Vgg19 utilizando las imágenes de variabilidad menor.

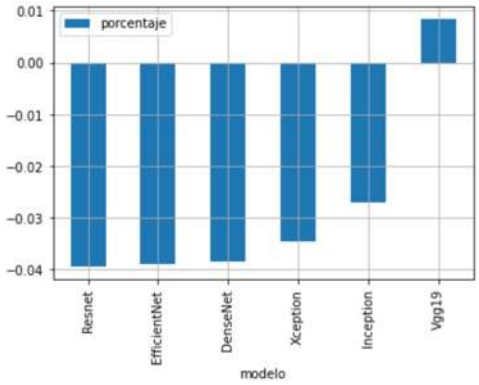

Figura 5.12: Gráfico de barras de acuerdo con la variabilidad de tamaño 9x9

En el gráfico 5.12, se observa que se obtuvo un mayor rendimiento en los modelos cuando se tenían imágenes con un bajo índice de variabilidad.

Dentro de todas las tablas de variabilidad mostradas se ha encontrado que todas tienen en común que los modelos tienden a tener un mejor rendimiento cuando se les pasa imágenes con un índice de variabilidad menor y tienden a tener un peor rendimiento cuando se usan imágenes con un índice de variabilidad mayor.

5.2.5. Reducir el índice de variabilidad en una imagen

Como se ha visto en la sección anterior , los modelos preentrenados se han comportado mejor cuando se le pasaban imágenes con una variabilidad de color menor, es decir que dentro de una área pequeña de píxeles hay variaciones de color mas pequeñas

Tomando en cuenta lo anterior se propuso una forma de hacer que las imágenes tengan una variabilidad de color menor antes de pasarlas al clasificador de imágenes, y esto se puede lograr usando filtros para suavizar las imágenes.

Se utilizo la técnica de suavizado de imágenes para lograr reducir el índice de variabilidad de color , y así ver que impacto tenia en las imágenes procesadas en la clasificación.

Filtrado de imágenes

Al igual que las señales unidimensionales, las imágenes también pueden ser filtradas con varios tipos de filtros, como por ejemplo, filtros pasa bajo (FPB), filtros pasa alto (FPA), filtros pasa banda, etc. Mientras que un FPB ayuda a eliminar el ruido en la imagen o a difuminar la imagen, un FPA ayuda a encontrar los bordes en una imagen. No obstante, también hay filtros que eliminan el ruido con poco efecto sobre los bordes.

Se utilizo el filtro bilateral ya que al ser aplicado , suaviza las imágenes pero remarcando los bordes dentro de las imágenes, los filtros de mediana y gaussiano son filtros con diferentes técnicas muy utilizados para el suavizado imágenes.

Suavizado del conjunto de imágenes

Al hacer uso de las funciones que vienen en la libreria OpenCv, se procesaron todas las imágenes del conjunto de datos, como se observa en la figura 5.13, se tomaron 3 imágenes del conjunto, y se muestra las diferencias al utilizar los diferentes filtros.

Figura 5.13: Suavizado del conjunto de imágenes

Como se muestra , hay diferencias notorias entre la imagen suavizada y la original para cada una de las imágenes mostradas

Índice de variabilidad de color sobre el conjunto de imágenes suavizado

Una vez obtenido el conjunto suavizado , visualmente si se aprecia que se suavizaron y quitamos el ruido de las imágenes , ahora se calcularon los índices de variabilidad de color para estas imágenes suavizadas.

En la siguiente figura se ha graficado la reducción que ha tenido cada una de las imágenes en el índice de variabilidad respecto a la imagen original

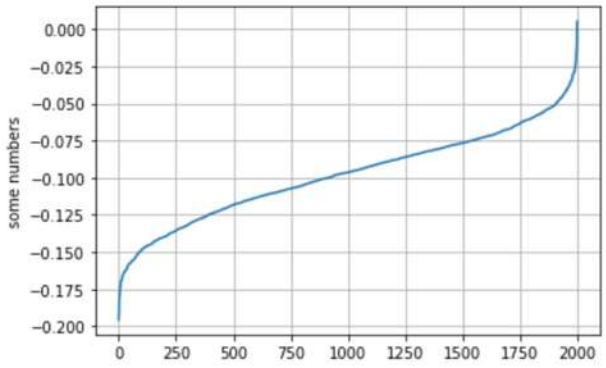

Figura 5.14: Gráfica de la reducción del índice de variabilidad

Se puede observar en la figura 5.14, que se ha reducido el índice de variabilidad, al rededor de un 10 % respecto a la original.

Evaluación de las imágenes suavizadas

Se procede a evaluar todo el conjunto de imágenes suavizadas con las diferentes redes convolucionales preentrenadas y así se obtiene la siguiente gráfica comparando la clasificación con las imágenes originales y con las imágenes suavizadas.

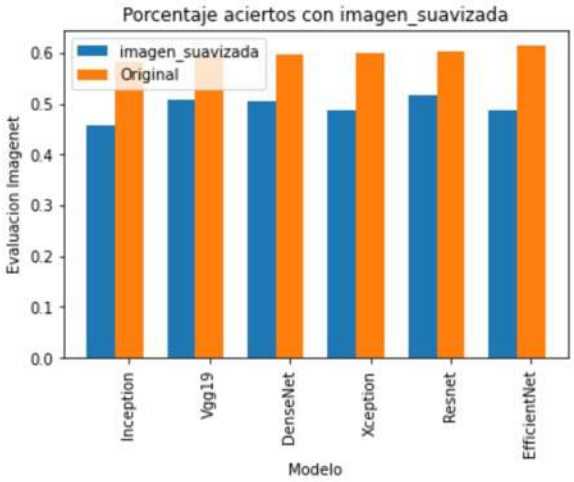

Figura 5.15: Gráfica de la reducción del índice de variabilidad

Al observar la comparación , es perceptible que bajo considerablemente el rendimiento al clasificar las imágenes respecto con las imágenes originales, lo que nos dice que aunque sea una modificación visualmente mejorable a las imágenes , tiene un impacto negativo en la clasificación de las mismas

6 Conclusiones

Se cumplió con el objetivo general y los objetivos específicos propuestos en el diseño y evaluación de redes neuronales convolucionales, al evaluar las redes neuronales convolucionales se ha visto que:

- Tras calcular el índice de luminosidad en las imágenes , todas las redes neuronales convolucionales funcionaron mejor al evaluarse con imágenes con alto índice , es decir con imágenes con mayor iluminación.
- En cuanto al índice de varianza , al ser evaluadas , se comportan mejor con las imágenes de alto índice de varianza , lo que indica que tienen un mejor comportamiento cuando las imágenes tienen mayor variedad de color y un peor comportamiento cuando se evalúan imágenes que son en su mayor parte del mismo color.
- Después de la propuesta del algoritmo para medir el nivel de variabilidad de color en una imagen a través de una área pequeña de píxeles, calcular el índice de variabilidad y dividir el subconjunto en alto y bajo índice , fue notorio que todas las redes neuronales siguieron un comportamiento , aun variando el tamaño del área de píxeles a tomar , fue claro que se comportaron mejor al utilizar el subconjunto con índice de variabilidad de color menor , y tuvo menor eficiencia al usar imágenes con alto índice de variabilidad de color , lo que indica que las imágenes con altos cambios de colores entre píxeles como puede ser una imagen con un fondo muy colorido y muchos cambios de color puede afectar negativamente a las redes neuronales convolucionales en cuanto a la clasificación de imágenes .

Al obtener los resultados esperados con el algoritmo propuesto del índice de variabilidad de color se propuso , bajar el índice de variabilidad en las imágenes con un alto índice de variabilidad utilizando el suavizado de imágenes.
Al evaluar las imágenes suavizadas se obtuvo que efectivamente se logro reducir el índice de variabilidad de color en las imágenes, y al evaluarlas de nuevo en la red neuronal convolucional se obtuvo una eficiencia menor a la obtenida con las imágenes antes de ser suavizadas, lo que nos podría indicar que aunque eran visualmente iguales , el haber hecho esta modificación en su estructura a las imágenes afecto negativamente al funcionamiento interno de las redes neuronales convolucionales.

Bibliografía

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

[2] A. C. M. y Sarah Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*, 2016.

[3] J. Cutler y M. Dickenson, «Introduction to Machine Learning with Python», en *Computational Frameworks for Political and Social Research with Python*, Springer, 2020, págs. 129-142.

[4] O. G. Yalçın. (2020). «4 Pre-Trained CNN Models to Use for Computer Vision with Transfer Learning», dirección: `https://towardsdatascience.com/4-pre-trained-cnn-models-to-use-for-computer-vision-with-transfer-learning-885cb1b2dfc`.

[5] R. Holbrook. (2020). «Intro to Deep Learning», dirección: `https://www.kaggle.com/learn/intro-to-deep-learning`.

[6] K. J. M. Ayuque Arenas. (2016). «Diseño de un sistema de clasificación de señales de tránsito vehicular utilizando redes neuronales convolucionales».

[7] O. G. Yalçın. (2020). «Funcion de brillo y contraste», dirección: `https://pro.arcgis.com/es/pro-app/latest/help/analysis/raster-functions/contrast-and-brightness-function.htm`.

[8] O. de Lama. (2015). «How to compute rgb image standard deviation from channels statistics», dirección: `https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics`.

[9] (2022). «Regresion polinomicas no lineales», dirección: `https://colab.research.google.com/github/jdamaster/machineLearningDiplomat/blob/master/s07_SVM__redes_elasticas_Regresion_polinomicas_no_lineales.ipynb`.

[10] J. B. Vilata. (2021). «Que es deep learning caso practico», dirección: `https://careers.edicomgroup.com/que-es-deep-learning-caso-practico-en-edicom-parte-i/`.

[11] (2021). «Inception v2 and v3», dirección: `https://www.geeksforgeeks.org/inception-v2-and-v3-inception-network-versions/`.

[12] A. Kaushik. (2020). «Understanding the vgg19 architecture», dirección: `https://iq.opengenus.org/vgg19-architecture/`.

[13] Z. Akhtar. (2021). «Xception: Deep learning with depth-wise separable convolutions», dirección: `https://iq.opengenus.org/xception-model/`.

[14] P. Mishra. (2019). «Convolutional neural networks (cnn)», dirección: `https://iq.opengenus.org/convolutional-neural-networks/`.

[15] A. Kaushik. (2020). «Understanding the resnet50 architecture», dirección: `https://iq.opengenus.org/resnet50-architecture/`.

[16] M. L. Tokio. (2020). «Cnn Architectures: Densenet», dirección: `https://machinelearningtokyo.com/2020/04/21/cnn-architectures-densenet/`.

[17] A. Nain. (2020). «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks», dirección: `https://programmerclick.com/article/57551464795/`.

[18] S. Bianco, R. Cadene, L. Celona y P. Napoletano, «Benchmark analysis of representative deep neural network architectures», *IEEE Access*, vol. 6, págs. 64 270-64 277, 2018.