UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

TESIS

DISEÑO Y EVALUACIÓN DE UN MODELO BASADO EN UNA RED DE CÁPSULAS DE MATRICES CON EM ROUTING UTILIZANDO UNA RED CONVOLUCIONAL DENSA

Para optar al título profesional de:
INGENIERO INFORMÁTICO Y DE SISTEMAS
Presentado por:
Br. BERNAL RIOS PAUL THEO
Br. BLAS HUAMAN WASHINGTON
Asesor :
M.Sc. ORMEÑO AYALA YESHICA ISELA
Co-asesor :
Ing. FARFÁN ESCOBEDO JEANFRANCO DAVID

CUSCO - PERÚ
2020

El presente trabajo está dedicado a Dios, por ser el inspirador y darme la fuerza para continuar. A mis padres Teodulfo y Estela, por su amor, trabajo y sacrificio en todos estos años, que gracias a ellos he logrado llegar hasta aquí y convertirme en lo que soy. A mi hermana Flor por estar siempre presente, acompañándome en todo momento y por brindarme fuerza en los momentos de flaqueza. También, está dedicado a la memoria de mis abuelos Paulino, Antonia, Julio y Victoria por el amor especial y el apoyo que me brindaron.
- Paul Theo Bernal Rios.

Dedico esta Tesis a mis padres Eulogio y Leonarda que siempre me apoyaron incondicionalmente en la parte moral y económica para poder llegar a ser un profesional. A mis hermanas, familiares y amigos por el apoyo que siempre me brindaron día a día en el transcurso de cada año de mi carrera.
- Washington Blas Huaman.

Agradecimientos

Agradecemos a Dios por bendecirnos la vida, por guiarnos a lo largo de nuestra existencia, ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad.
Gracias a nuestros padres: Teodufo y Estela; y, Eulogio y Leonarda, por ser los principales promotores de nuestros sueños, por confiar y creer en nuestras expectativas, por los consejos, valores y principios que nos han inculcado.
Agradecemos a nuestros docentes, por haber compartido sus conocimientos a lo largo de la preparación de nuestra profesion, de manera especial, a la docente Yeshica Isela Ormeño Ayala, asesora de nuestro proyecto de investigacion quien ha guiado con su paciencia, y su rectitud como docente.

Resumen

Actualmente la arquitectura de Aprendizaje Profundo *(Deep Learning)* con mayor éxito y más usado en reconocimiento y clasificación de imágenes, visión computacional, conducción automática de vehículos; es la Red Neuronal Convolucional *(Convolutional Neural Network o CNN)*. Sin embargo las *CNNs* presentan limitaciones ya que no son robustas cuando existen transformaciones en el objeto evaluado, es decir un ligero cambio de la posición del objeto provocaría que las *CNNs* modifiquen su predicción. Aunque este problema puede ser reducido incrementando el conjunto de datos durante el entrenamiento, esto no garantiza que la red sea robusta para una nueva modificación en la posición que pueda estar presente en el conjunto de datos de prueba. Para superar estos inconvenientes surgen las Redes de Cápsulas *(Capsule Network o CapsNet)*, que mejoran la clasificación de imágenes cuando estos presentan rotación, inclinación u otra orientación diferente, facilitando la obtención de información en la verificación de sus posiciones relativas con un menor número de datos. Por otro lado las redes de *Cápsulas Matriciales con Enrutamiento EM*, que son una mejora de las redes de *Cápsulas con Enrutamiento Dinámico*, presentan cápsulas donde cada cápsula está conformada por una unidad de activación que representan la presencia de un objeto y una matriz de pose de 4x4 que aprende a representar la relación espacial entre el objeto y el espectador, obteniendo así información más precisa. Ambas arquitecturas de redes de cápsulas presentan como primera capa de entrada una red de convolución estándar, pero un problema inherente a una *(CNN)* es cuando tiende a perder información a medida que la red se hace más profunda debido a la anulación del gradiente (vanishing gradient), es decir el gradiente tiende a ser igual a cero durante el proceso de aprendizaje cuando una red tiene muchas capas. Como una alternativa se tiene la arquitectura de Red Convolucional Densamente Conectado *(DenseNet)* que resuelve éste problema asegurando un flujo máximo de información, donde para cada capa, las características obtenidas en todas las capas anteriores se utilizan como entradas, y sus propias características obtenidas se utilizan como entradas en todas las capas posteriores, esto conduce a un mejor flujo de gradiente en comparación con las capas de convolución apiladas directamente. Por tanto, en este trabajo se diseña y evalúa una arquitectura de *Cápsulas Matriciales con Enrutamiento EM*, reemplazando su primera capa de red convolucional simple *(ReLU Conv1)* por una red convolucional densa *(DenseNet)*, para mostrar una mejora en el tiempo y precisión de entrenamiento frente al modelo base de redes de *Cápsulas Matriciales con Enrutamiento EM*, utilizando el conjunto de datos *SmallNORB* que está destinado a experimentos de reconocimiento de imágenes de objetos en 3D.

Palabras Claves: *Cápsulas Matriciales con Enrutamiento EM, CaspNet, DenseNet, Modelo.*

Abstract

Currently the Deep Learning architecture with greater success and more used in image recognition and classification, computational vision, automatic vehicle driving; it is the Convolutional Neural Network (CNN). However, the CNNs have limitations since they are not robust when there are transformations in the object evaluated, that is, a slight change in the position of the object would cause CNNs to modify their prediction. Although this problem can be reduced by increasing the data set during training, this does not guarantee that the network is robust for a new modification in the position that may be present in the test data set. To overcome these inconveniences, arise the Capsule Networks (CapsNet), which improve the classification of images when they have rotation, inclination or other different orientation, facilitating the obtaining of information in the verification of their relative positions with a lower number of data. On the other hand, the matrix capsule networks with EM routing, which are an improvement of the capsule networks with dynamic routing, have capsules where each capsule is made up of a unit that captures characteristics that represent the presence of an object and a pose matrix 4x4, that learns to represent the spatial relationship between the object and the viewer, thus obtaining more accurate information. Both capsule network architectures present a standard convolution network as the first input layer, but a problem inherent to one (CNN) is when it tends to lose information as the network it gets deeper due to the cancellation of the gradient (leakage gradient), that is, the gradient tends to be zero during the learning process when a network has many layers. As an alternative there is the Densely Connected Convolutional Network (DenseNet) architecture that solves this problem by ensuring a maximum flow of information, where for each layer, the characteristics obtained in all the previous layers are used as inputs, and their own characteristics obtained are used as inputs in all subsequent layers, this leads to a better gradient flow compared to the directly stacked convolution layers. Therefore, in this work an architecture of matrix capsules with EM routing is designed and evaluated, replacing its first simple convolutional network layer (ReLU Conv1) with a dense convolutional network (DenseNet), to show an improvement in the time and accuracy of training, front the base architecture of matrix capsule networks with EM routing, using the SmallNORB data set that is intended for 3D object image recognition experiments.

Keywords: *Matrix Capsules with EM Routing, CaspNet, DenseNet, Model.*

Índice general

Índice de figuras

Índice de tablas

Parte I

Generalidades

Capítulo 1

Aspectos Generales

1.1. Problema de Investigación

1.1.1. Descripción del Problema

Las redes neuronales convolucionales (CNNs) tienen un enorme éxito en el Aprendizaje Automático; en la actualidad muchos de los problemas más interesantes se atacan a través de estas técnicas. Si bien es cierto que las CNNs son buenas para detectar las características de un objeto; el mayor problema de estas redes neuronales es que no toman en cuenta las relaciones espaciales como traslación y rotación, que son cambios inherentes a un objeto. Además, si se quiere que la red neuronal convolucional pueda reconocer un objeto rotado en un ángulo, se tiene que entrenar utilizando las imágenes del objeto rotado a ese ángulo en particular; conllevando a un incremento en el número de datos y tiempo de entrenamiento. Existe una mejor arquitectura que las CNNs que suplen las limitaciones en la detección de las posiciones relativas de un objeto utilizando menor número de datos, al cual se denomina Red de Capsulas (Capsule Network o CapsNet), sin embargo, la pregunta es qué tan bien trabaja este modelo. La respuesta a esto es que no lo hace mejor que una red convolucional.
Si bien es cierto las cápsulas matriciales con enrutamiento EM trabajan mejor con imágenes donde los objetos presentan modificaciones en su postura, requiriendo menor número de datos, estos requieren mayor tiempo de entrenamiento debido a la complejidad de su arquitectura, puesto que en cada neurona se efectúan cálculos matriciales y no cálculos escalares como en las neuronas de las redes neuronales convolucionales. Por tanto, es necesario reducir esta complejidad computacional de la arquitectura de red de cápsulas matriciales con enrutamiento EM para lograr mejorar su tiempo y precisión de entrenamiento.

1.1.2. Formulación del Problema

- ¿Es posible mejorar el tiempo de entrenamiento del modelo de *Capsule Network* basado en Cápsulas Matriciales con Enrutamiento EM utilizando el modelo propuesto?.
- ¿Es posible mejorar la precisión de entrenamiento del modelo de *Capsule Network* basado en Cápsulas Matriciales con Enrutamiento EM utilizando el modelo propuesto?.

1.2. Antecedentes

Desde el momento en que se dieron a conocer las redes de cápsulas, se ha contribuido con una variedad de experimentos donde se pretenden demostrar, analizar y mejorar la complejidad computacional y su rendimiento respecto al estado de arte. A continuación se presenta un conjunto de trabajos relacionados que describen algunos de estos casos.

1. (Mukhometzianov y Carrillo, 2018), en su investigación evaluaron el rendimiento del algoritmo de CapsNet con enrutamiento dinámico. Para lo cual propusieron realizar una comparación de CapsNet con tres clasificadores conocidos *(Fisherfaces, LeNet y ResNet)*. En los experimentos se utilizó una *Core i7 extreme 7th generación, 16 GB RAM, Nvidia GeForce 1050 Ti Mobile* para el entrenamiento de los algoritmos clasificadores conocidos y *Google cloud, Intel Xeon, 5 Gb RAM, NVIDIA Tesla K80* para el entrenamiento de CapsNet, para los cuales se utilizaron *datasets* de imágenes de rostros, señales de tráfico y objetos cotidianos, obteniendo los siguientes resultados:

   - Utilizando el algoritmo ***Fisherfaces*** para el *dataset Yale Face*[1] con 5850 imágenes de rostros, se obtuvo un tiempo promedio de entrenamiento de 5 minutos con un 98.2 % de precisión y el algoritmo ***CapsNet*** alcanzó un tiempo promedio de entrenamiento de 24 horas con un 95.3 % de precisión.
   - Ejecutando el algoritmo ***Fisherfaces*** para el *dataset MIT CBCL (faces)* con 5240 imágenes de rostros, se obtuvo un tiempo promedio de entrenamiento de 1 minuto con un 98.3 % de precisión y el algoritmo ***CapsNet*** alcanzó un tiempo promedio de entrenamiento de 14 horas con un 99.87 % de precisión.
   - Haciendo uso del algoritmo ***LeNet*** para el *dataset BelgiumTS (traffic signs)*[2] con 7000 imágenes de señales de tráfico, se obtuvo un tiempo promedio de entrenamiento menor a 1 minuto con un 98.2 % de precisión y el algoritmo ***CapsNet*** alcanzó un tiempo promedio de entrenamiento de 16 horas con un 92 % de precisión en 40 épocas.
   - Utilizando el algoritmo ***Resnet 50*** para el *dataset CIFAR-100 (objects)* con 60000 imágenes de objetos cotidianos, se obtuvo un tiempo promedio de entrenamiento de 20 horas en 200 épocas con un 65.5 % de precisión y el algoritmo ***CapsNet*** alcanzó un tiempo promedio de entrenamiento de 18 horas con un 18 % de precisión en 35 épocas.

2. (Chauhan y cols., 2018), se han planteado como objetivo analizar la convergencia del modelo de *Cápsulas matriciales con enrutamiento EM* utilizando diferentes optimizadores como: Adam, Adadelta, Adagrad y Rmsprop; además de varios hiper parámetros como el número de canales en la primera capa de convolución **(A)**, el número de capas de cápsulas primarias **(B)**, el número de cápsulas en capas de convolución siguiente a la capa primaria **(C)**,

[1]Fuente: Yale Face Database B `https://computervisiononline.com/dataset/1105138686`
[2]Fuente: BelgiumTS Dataset `https://btsd.ethz.ch/shareddata/`

número de cápsulas en la última capa de convolución **(D)** y número de iteraciones de enrutamiento. También se analiza la convergencia de los modelos *CNN, CapsNet* respecto a las *Cápsulas matriciales con enrutamiento EM.* Para los experimentos se utilizaron *datasets* como *smallNORB, CIFAR10 y Tiny ImageNet.* Durante la implementación se encontró múltiples problemas como:

- **Sensibilidad a los hiper parámetros:** (Hinton y cols., 2018) no tienen publicado el código de *Cápsulas matriciales con enrutamiento EM.* Las implementaciones que se probaron sufrieron una severa sensibilidad a los hiper parámetros. La sensibilidad afectó la convergencia y la precisión general del modelo.
- **Costo Computacional:** Las CapsNets son computacionalmente costosas y lentas. Por tanto se redujo el número de épocas de entrenamiento.

3. (Xi y cols., 2017), se plantearon como objetivo encontrar un modelo basado en CapsNet el cual produzca un error de prueba óptimo. Para lo cual propusieron varios modelos que van desde apilar más capas de cápsula hasta probar diferentes parámetros, donde propusieron las siguientes variaciones: apilamiento de más capas de cápsulas, aumentar el número de cápsulas primarias, modificar la escala de pérdida en la reconstrucción, aumentar el número de capas convolucionales antes de la capa de cápsulas y utilizar una función de activación personalizada. Utilizando el dataset MNIST se llegó a la conclusión de que agregar una capa de convolución aumenta la precisión en un 0.41 % y el uso de un conjunto de 4 modelos aumenta la precisión en un 1.85 % en comparación con el modelo base de CapsNet.

4. (Phaye y cols., 2018) con el objetivo de mejorar la complejidad computacional de las redes de cápsulas plantean las redes de cápsulas densas (DCNet) y redes de cápsulas diversas (DCNet ++). Los dos marcos propuestos personalizan a CapsNet mediante la sustitución de las capas convolucionales estándar con capas convolucionales densamente conectadas. Esto ayuda a incorporar mapas de características aprendidas por diferentes capas para formar las cápsulas primarias. En esencia, DCNet, agrega una red de convolución más profunda, que conduce al aprendizaje de mapas de características discriminatorias. Además, DCNet ++ utiliza una arquitectura jerárquica para aprender cápsulas que representan información espacial de una manera fina a gruesa, lo que la hace más eficiente para el aprendizaje de datos complejos. Los experimentos sobre la tarea de clasificación de imágenes utilizando conjuntos de datos de referencia demuestran la eficacia de las arquitecturas propuestas. DCNet logra un rendimiento de vanguardia (99.75 %) en el conjunto de datos MNIST con una disminución de veinte veces en las iteraciones de entrenamiento total, en comparación con CapsNet convencional. Además, DCNet ++ funciona mejor que CapsNet en el conjunto de datos SVHN (96.90 %), y supera en un 0.31 % al conjunto de siete modelos CapsNet en CIFAR-10, con una reducción de siete veces en el número de parámetros.

1.3. Justificación

A través de investigaciones y experimentos revisados se encuentra que a medida que se va incrementando capas de convolución a la red de cápsulas, éste incrementa su precisión de entrenamiento; esto conlleva a pensar que a mayor número de capas convolucionales (CNN) que se agreguen a la red de capsulas, mejor será la precisión de entrenamiento; esta afirmación viene a ser hasta cierto punto una verdad. Sin embargo, en una CNN estándar, el aumento de más capas conduce a un problema de anulación del gradiente (vanishing gradient), generando pérdida de información de las características del objeto analizado. Este problema ha sido resuelto con una modificación de la arquitectura de red convolucional, que simplifica las conexiones de las capas iniciales a las capas posteriores, es decir que dada una capa de una red, este aprende las características de capas anteriores a través de una operación de concatenación y las características aprendidas en esta capa son utilizadas en todas las capas posteriores; esto implica un mejor flujo de la información con menor número de parámetros, permitiendo entrenar redes más profundas, esta red es denominada Red Convolucional Densamente Conectada (DenseNets). Las redes de cápsulas son computacionalmente complejas, es decir requieren mayor capacidad de cómputo debido a los cálculos vectoriales y/o matriciales que se efectúan al momento de su entrenamiento, conllevando a un mayor tiempo de entrenamiento; entonces si se desea agregar más capas convolucionales a una arquitectura compleja esto implicaría un incremento en el tiempo de entrenamiento de la red, lo que implica que hay una necesidad de mejorar la complejidad computacional de la red de cápsulas. Es por ello que una alternativa de solución es utilizar DenseNet, que tiene la capacidad de lograr un mejor flujo de información mediante la concatenación de características y con menor número de parámetros, para reemplazar la capa de convolución que presenta la red de cápsulas matriciales, permitiendo mejoras en el tiempo y precisión de entrenamiento. Así problemas como: complejidad y rendimiento (tiempo, precisión y perdida de entrenamiento) del modelo de red de cápsulas matriciales se mejorarían utilizando una DenseNet.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar y evaluar un modelo de aprendizaje profundo basado en una red de Cápsulas Matriciales con Enrutamiento EM utilizando una Red Convolucional Densa.

1.4.2. Objetivos Específicos

Dentro de los objetivos específicos se incluyen:

- Realizar un análisis de los modelos: Red Convolucional Densamente Conectado (DenseNet) y Cápsulas Matriciales con Enrutamiento EM.
- Diseñar un modelo de red de Cápsulas Matriciales con Enrutamiento EM utilizando una Red Convolucional Densa.

- Entrenar el modelo propuesto frente al modelo de red de Cápsulas Matriciales con Enrutamiento EM, siendo este último denominado como modelo base.
- Evaluar experimentalmente el tiempo, precisión y pérdida de entrenamiento del modelo propuesto; para posteriormente analizar los resultados.

1.5. Alcances y Limitaciones

Para mejor entendimiento de la propuesta del modelo se limita la investigación a:

- Dado que el aporte del trabajo se ajusta a realizar entrenamiento del modelo, no se realizará el proceso de tratamiento de imágenes.
- Para el entrenamiento se utiliza el conjunto de datos SmallNORB que muestra imágenes de juguetes desde varios ángulos y con diferentes niveles de iluminación.
- Para el desarrollo de la implementación del modelo se hace uso exclusivo de una GPU NVIDIA 1070.
- La implementación del modelo propuesto frente al modelo de red de *Cápsulas Matriciales con Enrutamiento EM*, sólo será analizado en tiempo, precisión y pérdida de entrenamiento.

1.6. Metodología

1.6.1. Tipo de investigación.

El tipo de investigación realizado es exploratoria y experimental.

1.6.2. Diseño metodológico.

El diseño de la metodología a seguir para el desarrollo de la presente investigación está basada en las siguientes fases esquematizados en la Figura 1.1.:

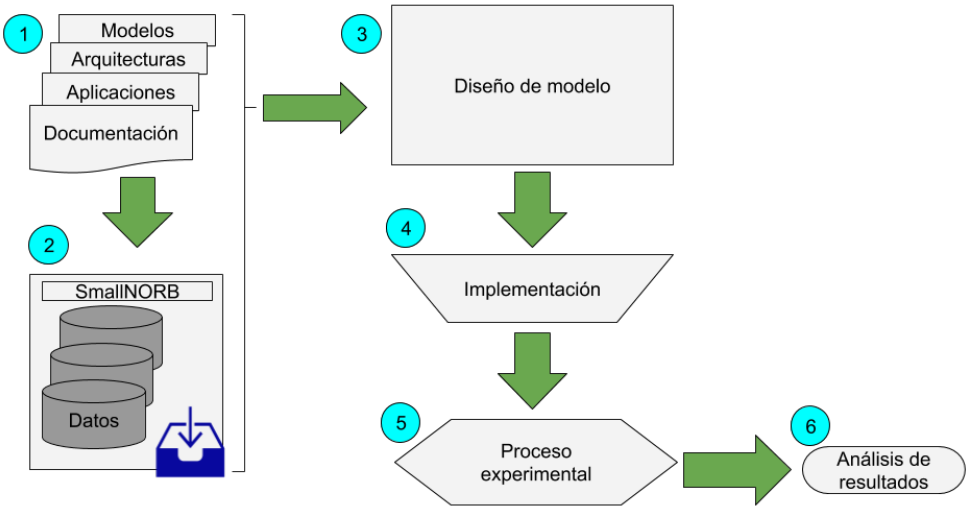

Figura 1.1: Esquema de la metodología del modelo propuesto.
Fuente: Propio

1. Revisión bibliográfica relacionado al aprendizaje profundo como modelos y arquitecturas, experimentos, aplicaciones, clasificación de imágenes, redes neuronales capsulares, es decir, información necesaria y relacionada al modelo propuesto.

2. Obtención de dataset, que habiendo realizado la revisión bibliográfica pertinente, se realiza la descarga del conjunto de datos SmallNORB para su entrenamiento, habiendo sido estos datos seleccionado por su aplicación experimental en trabajos de investigación basados en redes de cápsulas (Hinton y cols., 2018) (Phaye y cols., 2018).

3. Diseño del modelo propuesto, donde se modifica la red de *Cápsulas Matriciales con Enrutamiento EM*, reemplazando su primera capa de *Red Convolucional Estándar* por una *Red Convolucional Densa*.

4. Implementación de los modelos (*modelo propuesto y modelo de red de Cápsulas Matriciales con Enrutamiento EM*), utilizando herramientas y tecnologías, seleccionados pertinentemente.

5. Proceso experimental que consiste en definir el entrenamiento basado en el modelo propuesto y el modelo base (*Red de Cápsulas Matriciales con Enrutamiento EM*).

6. Análisis de los resultados obtenidos que muestra el tiempo y precisión (accuracy) de entrenamiento del modelo propuesto y el modelo base.

1.7. Cronograma de Actividades

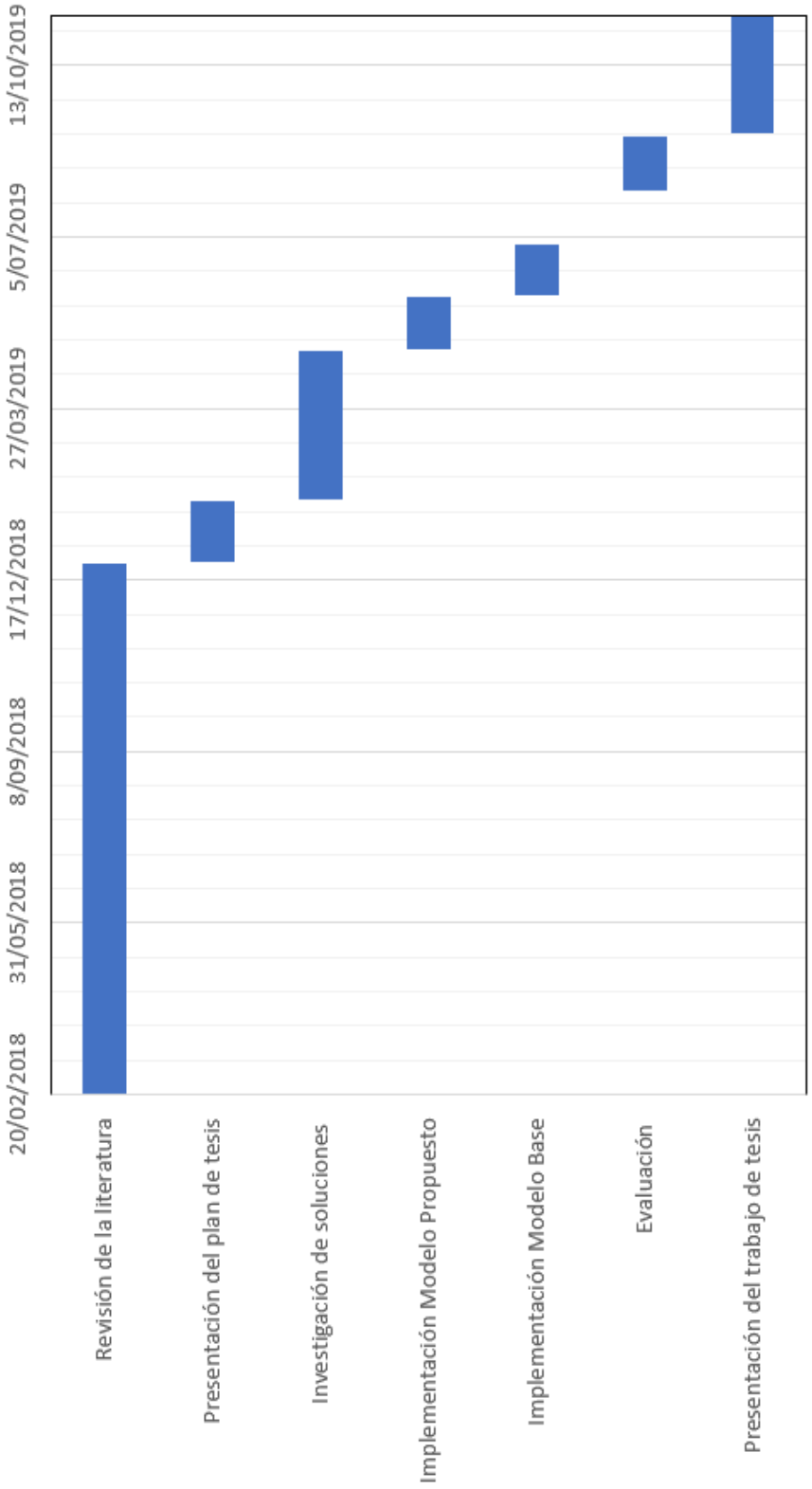

Figura 1.2: Cronograma de Actividades.
Fuente: Fuente propia

Parte II

Marco Teórico

Capítulo 2

Marco conceptual

2.1. Visión Computacional

La experiencia en el mundo en que vivimos está cubierta por una variedad sin fin de objetos, animados e inanimados. Así pues, si la visión es un medio para un fin "conocer el mundo observándose". La visión computacional es exactamente lo mismo salvo que el medio por el cual se adquiere el conocimiento ahora es un instrumento de cómputo. El principal beneficio de la visión computacional es la alta precisión con la que puede reemplazar la visión humana si se entrena correctamente. Los seres humanos, usan los ojos y el cerebro para analizar un entorno visual. Esto para los humanos es natural y lo hace bastante bien. Una computadora, por otro lado, no puede hacer esto automáticamente, necesita algoritmos y datos para aprender lo que está "viendo". Requiere mucho esfuerzo, pero una vez que una computadora aprende a hacerlo, puede hacerlo mejor que casi cualquier ser humano. Esto puede hacer que los procesos sean más rápidos y simples al reemplazar cualquier actividad visual. A diferencia de los humanos, que pueden sentirse abrumados o sesgados, una computadora puede ver muchas cosas a la vez, con gran detalle, y analizarlas sin cansarse. La precisión del análisis por computadora puede generar enormes ahorros de tiempo y mejoras de calidad, y por lo tanto liberar recursos que requieren interacción humana.

2.2. Aprendizaje de Máquina

El aprendizaje de máquina es un método de análisis de datos que automatiza la construcción de modelos analíticos. Es una rama de la inteligencia artificial basada en la idea de que los sistemas pueden aprender de datos, identificar patrones y tomar decisiones con mínima intervención humana.
Existen tres tipos principales de Aprendizaje Automático:

- **Aprendizaje Supervisado.-** Este tipo de aprendizaje se basa en lo que se conoce como información de entrenamiento. Se entrena al sistema proporcionándole cierta cantidad de datos definiéndose al detalle con etiquetas. Por ejemplo, proporcionando a la computadora fotos de perros y gatos con etiquetas que los definen como tales. Una vez que se le ha proporcionado la suficiente cantidad de dichos datos, podrán introducirse nuevos datos sin necesidad de etiquetas, en base a patrones distintos que ha venido registrando

durante el entrenamiento.

- **Aprendizaje no Supervisado.-** En este tipo de aprendizaje no se usan valores verdaderos o etiquetas. Estos sistemas tienen como finalidad la comprensión y abstracción de patrones de información de manera directa. Este es un modelo de problema que se conoce como clustering. Es un método de entrenamiento más parecido al modo en que los humanos procesan la información.

- **Aprendizaje por Refuerzo.-** En este tipo de aprendizaje los sistemas aprenden a partir de la experiencia. Es una técnica basada en la prueba y error, y en el uso de funciones de premio que optimizan el comportamiento del sistema. Es una de las maneras más interesantes de aprendizaje para sistemas de Inteligencia Artificial, pues no requiere de la introducción de gran cantidad de información.

2.3. Redes Neuronales

2.3.1. Redes Neuronales Artificiales

Las redes neuronales artificiales son un modelo matemático formal que está inspirado en la operación biológica de las neuronas en el cerebro humano, en el sentido de integrar una serie de entradas y proporcionar cierta respuesta, que se propaga por el axón (Martín del Brío y Sanz, 2002). Una red de neuronas artificiales hace referencia a las interconexiones entre neuronas en las diferentes capas que presenta el modelo. Dado una red neuronal de tres capas, la primera capa presenta neuronas de entrada que envían datos a través de las sinapsis a la segunda capa de neuronas, y luego a través de más sinapsis a la tercera capa de neuronas de salida. El proceso de sinapsis almacena parámetros llamados *pesos* que son aprendidos en el proceso de entrenamiento de una red neuronal(Agatonovic-Kustrin y Beresford, 2000).

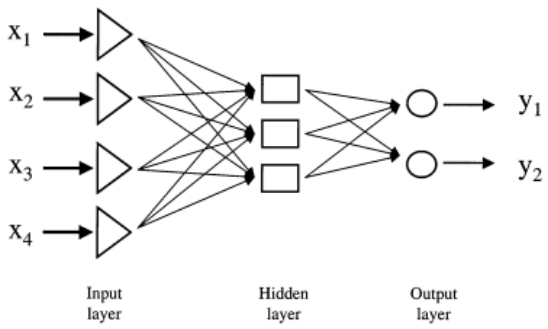

Figura 2.1: Modelo de Red Neuronal Artificial de tres capas.
Fuente: (Agatonovic-Kustrin y Beresford, 2000)

2.3.1.1. Estructura base de una neurona artificial

La estructura base de una neurona sirve para modelar correctamente el comportamiento de toda una red neuronal. Donde, dado una neurona y_j, ésta recibe

información de las n neuronas de entrada x_i. Los valores w_{ji} representan los *pesos* sinápticos en las dendritas de y_j. Note que i denota a la neurona hacia dónde se dirige la información y j denota de qué neurona procede la información.

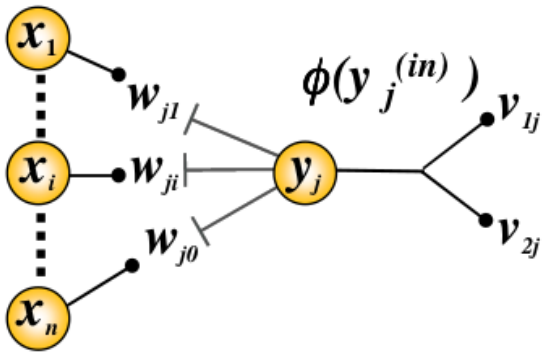

Figura 2.2: Estructura base de una neurona artificial.
Fuente: (Izaurieta y Saavedra, 2000)

Cada peso sináptico w_{ji} simplemente multiplica a su neurona de entrada x_i correspondiente. La neurona y_j resulta de aplicar una *función de activación* ϕ a $y_j^{(in)}$, el cual viene a ser la sumatoria de los productos entre la entrada con su respectivo peso $(w_{ji} * x_i)$ provenientes de todas las dendritas (Izaurieta y Saavedra, 2000).

$$\sum_{i=1}^{n}(w_{ji} * x_i) = y_j^{(in)} \tag{2.1}$$

$$y_j = \phi(y_j^{(in)}) \tag{2.2}$$

2.3.2. Función de activación

El valor de una neurona se obtiene a partir de transformar un valor de entrada mediante una función de activación no lineal ϕ.
Las funciones de activación no lineales se describen a continuación:

- **Sigmoide.-** Presenta líneas de saturación en el rango: $[0, 1]$.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

- **Tanh.-** Presenta líneas de saturación en el rango: $[-1, 1]$.

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.4}$$

- **Rectified Linear Unit (RELU).-** No presenta líneas de saturación, tiene un comportamiento lineal para entradas positivas, siendo su rango: $[0, x]$.

$$RELU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.5}$$

Al utilizar funciones de activación que presentan líneas de saturación, los valores de la salida de las neuronas, y durante ciertas fases del proceso de entrenamiento de la red neuronal no cambian significativamente, lo que hace que el gradiente no cambie y por tanto los parámetros de la red no sufren cambios apreciables, lo que genera un estancamiento en el proceso de entrenamiento. Una ventaja de utilizar la función RELU es que al no poseer estas regiones de saturación evita un estancamiento durante el proceso de entrenamiento.

2.3.3. Aprendizaje

El aprendizaje en una red neuronal, es un proceso en el cual ocurren cambios, esto implica la destrucción, modificación y creación de conexiones neuronales. En una red neuronal artificial el proceso de aprendizaje tiene como objetivo ajustar los parámetros de la red (pesos y umbrales), con el fin de que las entradas presentadas produzcan las salidas deseadas, es decir, con el fin de minimizar la función error o de costo Isasi Vinuela y Galván León (2004).

2.3.3.1. Algoritmo de RetroPropagación

El algoritmo de Retropropagación (Backpropagation), es una técnica de aprendizaje supervisado mediante el cual se van adaptando y modificando todos los parámetros de la red durante el descenso de gradiente, es decir, la modificación de los parámetros se realiza para que la salida de la red sea lo más próxima posible a la salida proporcionada por el supervisor o salida deseada. Por tanto, para cada patrón de entrada a la red es necesario disponer de un patrón de salida deseada. El descenso de gradiente es un método que permite encontrar los pesos y umbrales que minimicen la función de costo. La función de error o de costo evalúa la diferencia entre las salidas de la red y las salidas deseadas.
La función de error se define como:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n) \tag{2.6}$$

Donde: N es número de parámetros o muestras y $e(n)$ es el error cometido por la red para el patrón n, dado por:

$$e(n) = \frac{1}{2}\sum_{i=1}^{n_C}(s_i(n) - y_i(n))^2 \tag{2.7}$$

siendo $Y(n) = (y_1(n), ..., y_{n_C}(n))$ y $S(n) = (s_1(n), ..., s_{n_C}(n))$ los vectores de salidas de la red y salidas deseadas para el patrón n, respectivamente Isasi Vinuela y Galván León (2004).

2.4. Deep Learning

Deep learning es un tipo de machine learning que entrena a una computadora para que realice tareas como las hace los seres humanos, como el reconocimiento del habla, la identificación de imágenes o hacer predicciones. En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, deep learning

configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento.

2.4.1. Red Neuronal Convolucional (CNN)

Las redes neuronales convolucionales (CNN) es un tipo de red neuronal artificial con aprendizaje supervisado que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos y "ver". Una CNN consiste de varias capas, las principales son convolución y reducción de muestreo. Cada una recibe un dato de entrada en 3 dimensiones (ancho, alto y profundidad) que es transformado en dato de salida en 3 dimensiones diferentes.

2.4.1.1. Convolución

Una convolución es un operador matemático que trabaja con dos argumentos y las transforma en una tercera, (Pacheco, 2017). En procesamiento de imágenes la operación de convolución recibe como entrada o input la imagen, el cual se mapea en una matriz de pixeles y luego aplica sobre ella un filtro o kernel que nos devuelve un mapa de las características de la imagen original, de esta forma se logra reducir el tamaño de los parámetros. La convolución aprovecha tres ideas importantes que pueden ayudar a mejorar cualquier sistema de machine learning, ellas son:

- **Interacciones dispersas**, ya que al aplicar un filtro de menor tamaño sobre la entrada original se puede reducir drásticamente la cantidad de parámetros y cálculos.
- **Los parámetros compartidos**, que hace referencia a compartir los parámetros entre los distintos tipos de filtros, ayudando también a mejorar la eficiencia del sistema.
- **Las representaciones equivariantes**, que indican que, si las entradas cambian, las salidas van a cambiar también en forma similar.

La convolución proporciona un medio para trabajar con entradas de tamaño variable, lo que puede ser también muy conveniente.

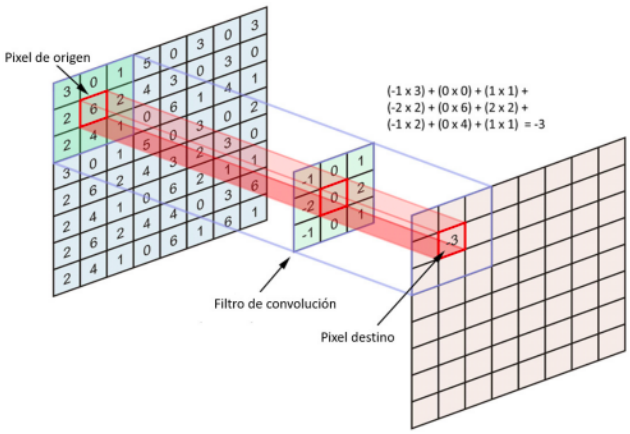

Figura 2.3: Obtención básica de convolución.
Fuente: Pacheco (2017)

En cada operación de convolución se desliza el kernel a la derecha sobre la imagen con un desplazamiento o padding igual a uno y se vuelve a calcular la convolución, almacenando el resultado en la siguiente posición de la matriz de activación. De esta manera, este proceso iterativo se repite a lo largo de toda la imagen desplazándose de izquierda a derecha y bajando una unidad al llegar a un borde. Una vez que se ha recorrido toda la imagen se obtiene la matriz de activación completa que contiene las características que se buscan en la imagen para cada filtro.

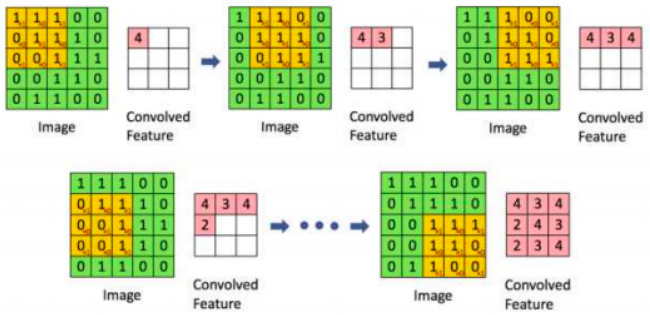

Figura 2.4: Obtención de matriz de características.
Fuente: Pacheco (2017)

2.4.1.2. Capa de reducción o Pooling

La capa de reducción o pooling se coloca generalmente después de la capa convolucional. Su utilidad principal radica en la reducción de las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa convolucional. No afecta a la dimensión de profundidad del volumen. La operación realizada por esta capa también se llama reducción de muestreo, ya que la reducción de tamaño conduce también a la pérdida de información. Sin embargo, una pérdida de este tipo puede ser beneficioso para la red por dos razones:

- La disminución en el tamaño conduce a una menor sobrecarga de cálculo para las próximas capas de la red.

- También trabaja para reducir el sobreajuste.

La operación que se suele utilizar en esta capa es max-pooling, que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se va quedando con el máximo valor.

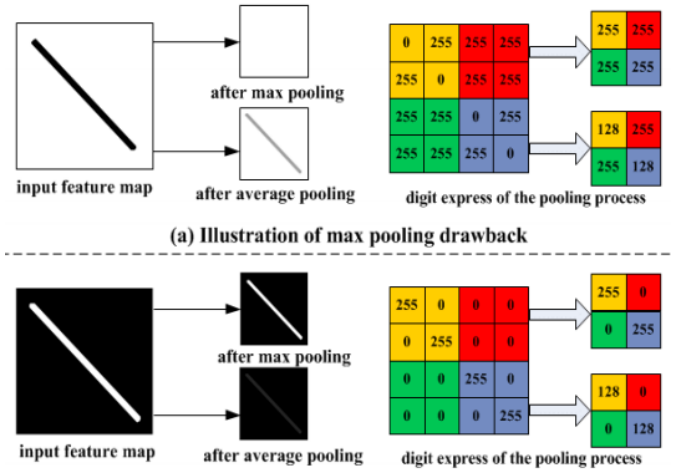

Figura 2.5: Resultado de aplicar max pooling y average pooling.
Fuente: Pacheco (2017)

2.4.1.3. Capa totalmente conectada o Fully-Connected

Al final de las capas convolucional y de pooling, las redes utilizan generalmente capas completamente conectados en la que cada píxel se considera como una neurona separada al igual que en una red neuronal regular. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se debe predecir. La capa Fully-Connected es un Perceptrón Multicapa (MLP) que utiliza una función de activación Softmax en la capa de salida. Sin embargo, también se pueden utilizar otros clasificadores. La capa Fully-Connected implica que cada neurona en la capa anterior está conectada a cada neurona en la siguiente capa.

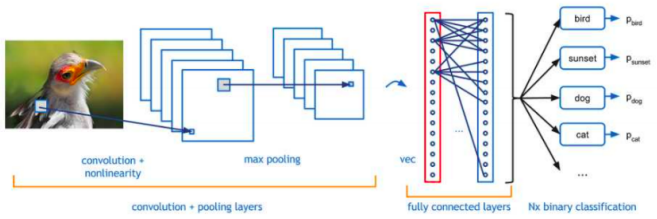

Figura 2.6: Esquema de una red neuronal convolucional.
Fuente: Pacheco (2017)

2.4.2. Red Convolucional Densamente Conectado (DenseNet)

La red convolucional densa (DenseNet) se basa en concatenar cada salida de las capas previas hacia las siguientes, es decir cada capa toma como entrada todos los valores de características anteriores(Gao Huang, 2016).

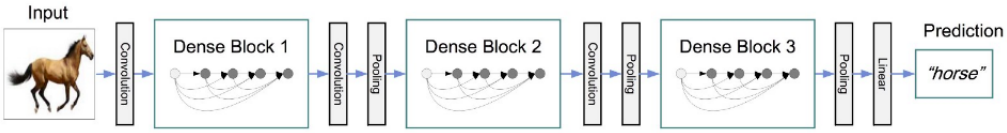

Figura 2.7: DenseNet con tres bloques densos.
Fuente: (Gao Huang, 2016)

En la Figura 2.3. se representa una red de tres bloques densos, las capas entre dos bloques adyacentes hace referencia a una capa de transición cambiando el tamaño del mapa de características mediante convolución y pooling.
A continuación se describe la estructura de una red convolucional densamente conectada:

2.4.2.1. Red densa

Sea X_0 un dato de entrada, el cual es pasado a través de una red convolucional que comprende L capas, cada una de las cuales implementa una transformación no lineal $H_l(.)$, donde l es el índice de la capa. $H_l(.)$ es una función compuesta de operaciones como Batch Normalization (BN), Rectified Linear Units (RELU), Pooling o Convolución (Conv). Se denota la salida de la capa $l_{ésima}$ como X_l(Gao Huang, 2016).

2.4.2.2. Conectividad densa

Para mejorar aún más el flujo de información entre capas, se introducen conexiones directas desde cualquier capa anterior a todas las capas posteriores. Por tanto, la capa $l_{ésima}$ recibe las características de todas las capas anteriores, $X_0, \ldots, X_{l-1}$, como entrada: $X_l = H_l([X_0, X_1, \ldots, X_{l-1]})$, donde $[X_0, X_1, \ldots, X_{l-1}]$ hace referencia a la concatenación de las características producidos en las capas $0, \ldots, l-1$. Debido a su densa conectividad, se denomina a esta arquitectura de red como *Dense Convolutional Network (DenseNet)*. Para facilitar la implementación, se concatenan las entradas múltiples de $H_l(.)$ en la ecuación $X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$ en un solo tensor(Gao Huang, 2016).

2.4.2.3. Función compuesta.

Se define $H_l(.)$ como función compuesta de tres operaciones consecutivas: Batch Normalization (BN), seguida por Rectified Linear Unit (RELU) y una Convolucion de 3x3 (Conv).

2.4.2.4. Capas de pooling

La operación de concatenación utilizada en la ecuación $X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$ no es viable cuando cambia el tamaño del tensor de características. Sin embargo, una parte esencial de las redes convolucionales son las capas *down-sampling* que cambian el tamaño del vector de características. Para facilitar el *down-sampling* se divide la red en múltiples bloques densos densamente conectados. Las capas entre bloques adyacentes son denominados capas de transición, en el cual se realizan las operaciones de convolución y pooling.

2.4.2.5. Tasa de crecimiento

Si cada función compuesta H_l produce K mapas de características, se deduce que la $l_{ésima}$ capa tiene K_0+K_{l-1} características de entrada, donde K_0 es el número de canales en la capa de entrada. Una diferencia importante entre DenseNet y las arquitecturas de red existentes es que DenseNet puede tener capas muy reducidos, por ejemplo, $K = 12$. Se hace referencia al hiper parámetro K como la tasa de crecimiento (*growth rate*) de la red. Cada capa agrega K características propios. La tasa de crecimiento regula la cantidad de información nueva que cada capa contribuye al estado global. Se puede acceder al estado global, una vez inscrito, desde cualquier lugar dentro de la red y, a diferencia de las arquitecturas de red tradicionales, no es necesario replicarlo de una capa a otra(Gao Huang, 2016).

2.5. Redes de Cápsulas

2.5.1. Cápsulas

Geoffrey E Hinton y Frosst (2017) explican que una cápsula es un grupo de neuronas que no sólo capturan la probabilidad, sino también los parámetros de las características. Así que la cápsula no solo detecta una característica, sino que está capacitada para aprender y detectar las variantes.
Las cápsulas representan las diversas características de una entidad particular que están presentes en la imagen. Una característica muy especial es la existencia de la entidad instanciada en la imagen. La entidad instanciada es un parámetro como posición, tamaño, orientación, deformación, velocidad, albedo, matiz, textura, etc. Una forma obvia de representar su existencia es mediante el uso de una unidad logística separada, cuya salida es la probabilidad de que la entidad exista. Para obtener mejores resultados que las CNN, se debe usar un mecanismo iterativo de enrutamiento por acuerdo. Estas características se denominan parámetros de instanciación.

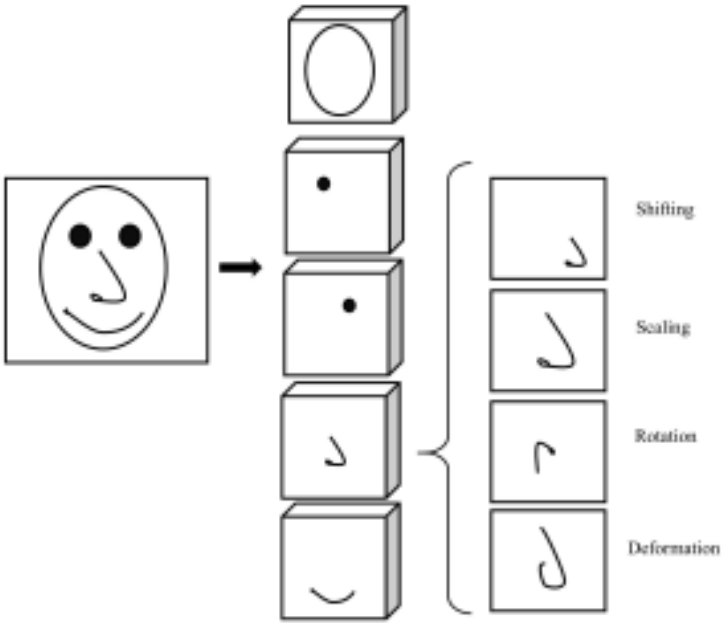

Figura 2.8: Ejemplo de parámetros de instanciación para una cápsula nasal en un problema de detección de rostros.
Fuente: (Shahroudnejad y cols., 2018)

2.5.2. Enrutamiento dinámico entre cápsulas.

2.5.2.1. Equivarianza

Conceptualmente el modelo CNN usa múltiples neuronas y capas para capturar diferentes variantes de características.

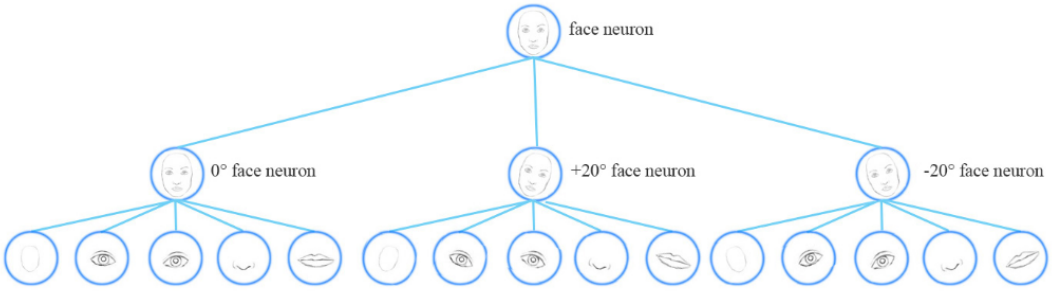

Figura 2.9: Ejemplo de captura de variantes de un rostro utilizando una red convolucional.
Fuente: https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/4

Una red de capsulas comparte la misma cápsula para detectar múltiples variantes en una red más simple.

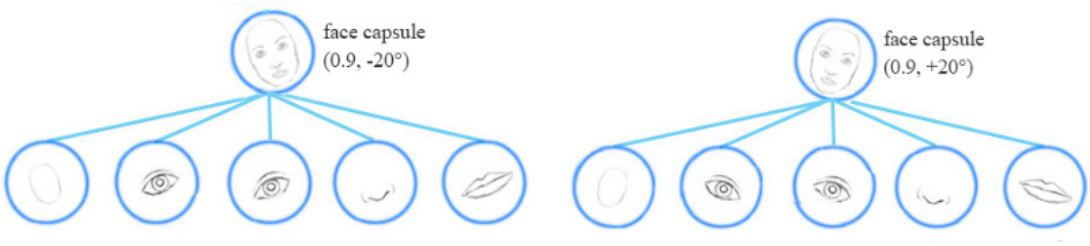

Figura 2.10: Ejemplo de captura de variantes de un rostro utilizando cápsulas.
Fuente: https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

Entonces se dice que equivarianza es la detección de objetos que pueden transformarse entre sí. Tomando como ejemplo la figura 2.5 se dice que intuitivamente una cápsula detecta que la cara se gira hacia la derecha 20° o a la izquierda 20° en lugar de darse cuenta de que la cara coincide con una variante que gira hacia la derecha 20°. Al forzar al modelo a aprender la variante de la característica en una cápsula, se puede extrapolar las variantes con menos datos de entrenamiento(Geoffrey E Hinton y Frosst, 2017).

2.5.2.2. Función Squash

En las redes neuronales profundas, las funciones de activación son operaciones matemáticas simples aplicadas a la salida de capas. Se utilizan para aproximar relaciones no lineales que existen en los datos. Las capas de activación generalmente actúan sobre valores escalares, por ejemplo, normalizando cada elemento en un vector para que caiga entre 0 y 1. En Capsule Networks, se utiliza un tipo especial de función de activación llamada función de squash para normalizar la magnitud de los vectores, en lugar de los propios elementos escalares. Los resultados de estas funciones de squash indican de cómo enrutar los datos a través de varias cápsulas que están capacitadas para aprender diferentes conceptos. Las propiedades de cada objeto en la imagen se expresan en los vectores que los enrutan. Por ejemplo, las activaciones de una cara pueden dirigir diferentes partes de una imagen a cápsulas que comprenden los ojos, la nariz, la boca y las orejas(Geoffrey E Hinton y Frosst, 2017).

$$V_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{2.8}$$

2.5.2.3. Enrutamiento dinámico

En este proceso de enrutamiento dinámico, las cápsulas de nivel inferior envían su entrada a las cápsulas de nivel superior que concuerdan con su entrada. Para cada cápsula superior a la que se puede enrutar, la cápsula inferior calcula un vector de predicción multiplicando su propia salida por una matriz de ponderación. Si el vector de predicción tiene un producto escalar grande con la salida de una posible cápsula más alta, hay una retroalimentación de arriba hacia abajo que tiene el efecto de aumentar el coeficiente de acoplamiento para esas cápsulas de alto nivel y disminuirlo para otras.

2.5.2.4. Enrutamiento iterativo dinámico

En la cápsula, se usa el enrutamiento dinámico iterativo para calcular la salida de la cápsula mediante el cálculo de un valor intermedio c_{ij} (coeficiente de acoplamiento).

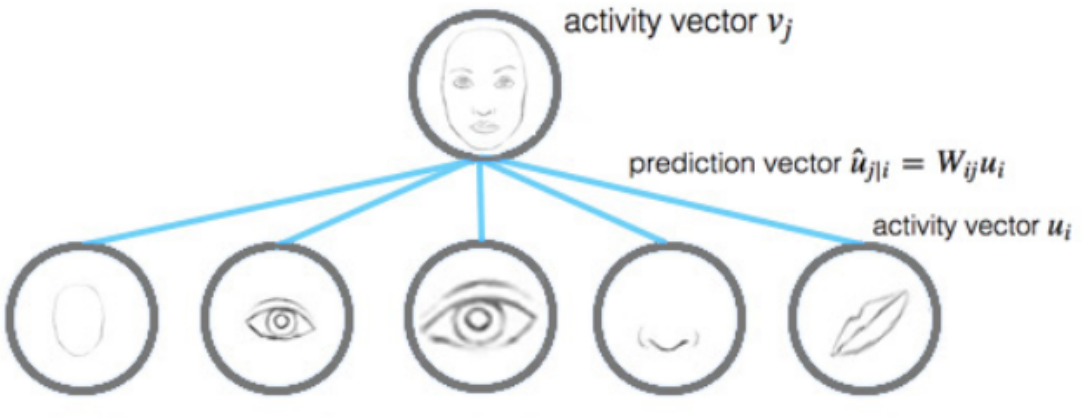

Figura 2.11: Manejo del enrutamiento iterativo dinámico en una cápsula.
Fuente: https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

El vector de predicción $\hat{\mu}_{j|i}$ se calcula como:

$$\hat{\mu}_{j|i} = W_{ij}\mu_i \tag{2.9}$$

Siendo Wij una matriz de transformación y u_i la entrada de una cápsula.
El vector de actividad V_j (cápsula de salida j) se calcula como:

$$s_j = \sum_i c_{ij}\hat{\mu}_{j|i} \tag{2.10}$$

$$V_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{2.11}$$

Intuitivamente, vector de predicción. $\hat{\mu}_{j|i}$ es la predicción (voto) de la cápsula i en la salida de la cápsula j, si el vector de actividad tiene una gran similitud con el vector de predicción, se llega a la conclusión de que ambas cápsulas están altamente relacionadas. Dicha similitud se mide utilizando el producto escalar de la predicción y el vector de actividad.

$$b_{ij} \leftarrow \hat{\mu}_{j|i} \cdot V_j \tag{2.12}$$

Por lo tanto, la puntuación de similitud b_{ij} toma en cuenta tanto la probabilidad como las propiedades de la característica, en lugar de la probabilidad en las neuronas. También, b_{ij} permanece bajo si la activación μ_i de cápsula i es bajo desde $\hat{\mu}_{j|i}$ la longitud es proporcional a μ_i es decir b_{ij} debe permanecer bajo entre la cápsula de la boca y la cápsula facial si la cápsula de la boca no está activada.
Los coeficientes de acoplamiento. c_{ij} se calcula como el softmax de b_{ij}:

$$c_{ij} = \frac{expb_{ij}}{\sum_k expb_{ik}} \tag{2.13}$$

Para hacer b_{ij} más preciso, se actualiza de forma iterativa en múltiples iteraciones (generalmente en 3 iteraciones)(Geoffrey E Hinton y Frosst, 2017).

$$b_{ij} \leftarrow b_{ij} + \hat{\mu}_{j|i} \cdot V_j \tag{2.14}$$

2.5.3. Cápsulas matriciales con enrutamiento EM

2.5.3.1. Cápsula Matricial

Una cápsula matricial está representada por una activación que hace referencia a la característica del objeto y una matriz de pose de 4x4 que define la orientación y posición del objeto. Esta estructura de una cápsula matricial equivale a ver al objeto desde diferentes posiciones.

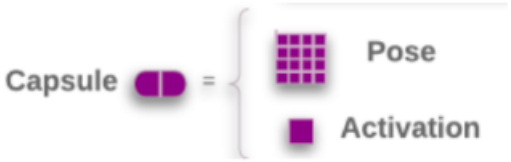

Figura 2.12: Estructura de una cápsula.
Fuente: (Hui, 2017)

El objetivo del enrutamiento EM, es agrupar las cápsulas para formar una relación parte-todo usando una técnica de agrupamiento denominado Algoritmo EM.

2.5.3.2. Algoritmo EM

El algoritmo EM (Garre y cols., 2007), es un método de agrupamiento probabilístico; éste se utiliza frecuentemente en aprendizaje automático y visión artificial para aprender *Modelos ocultos de Márkov* y *Distribuciones Gaussianas*, utilizados en procesos de clasificación o reconocimiento de imágenes.
El algoritmo EM alterna pasos de esperanza (E-step), donde se computará la esperanza[1] de la verosimilitud mediante la inclusión de variables latentes como si fueran observables, y un paso de maximización (M-step), donde se computan estimadores de máxima verosimilitud de los parámetros mediante la maximización de la verosimilitud esperada del paso E. Los parámetros que se encuentran en el paso M se usan para comenzar el paso E siguiente, y así el proceso se repite.
También se puede decir que el algoritmo EM agrupa puntos de datos en una mezcla de distribuciones gaussianas descritas por una media μ y una desviación estándar σ. Por ejemplo si agrupamos puntos de datos en grupos de color amarillo y rojo, donde cada uno es representada aleatoriamente con una media μ y una desviación σ.

[1]En estadística la esperanza matemática (también llamada esperanza, valor esperado, media poblacional o media) de una variable aleatoria X, es el número que formaliza la idea de valor medio de un fenómeno aleatorio.

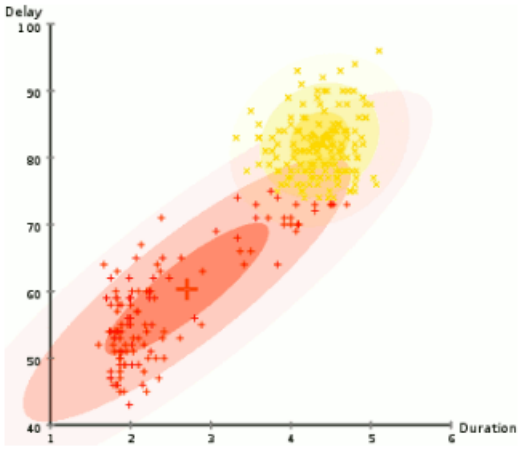

Figura 2.13: Puntos de datos agrupados de color rojo y amarillo.
Fuente: (Hui, 2017)

Dado que se tiene dos grupos (*amarillo y rojo*), se inicializa aleatoriamente los grupos en $G_1 = (\mu_1, {\sigma_1}^2)$ y $G_2 = (\mu_2, {\sigma_2}^2)$ respectivamente. El algoritmo EM intenta ajustar los puntos de los datos de entrenamiento en G_1 y G_2 y luego vuelve a calcular μ y σ para G_1 y G_2 en función de la distribución gaussiana. La iteración continúa hasta que la solución converge en G_1 y G_2; ésto es cuando la probabilidad de ver todos los puntos de datos agrupados convenientemente en G_1 y G_2 sea lo máximo posible.
La probabilidad de que un dato x pertenezca al grupo G_1 es:

$$P(x|G_1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \tag{2.15}$$

2.5.3.3. Enrutamiento EM

En un objeto dado, cada una de las cápsulas en la capa inferior hace predicciones (*votos*) sobre las matrices de pose de sus posibles cápsulas parentales. Cada voto es un valor predicho para la matriz de pose de una cápsula principal, y se calcula multiplicando la propia matriz de pose M con una matriz de transformación W, donde éste último es aprendido durante el entrenamiento del conjunto de datos de entrenamiento.
En la Figura 2.15. se aplica enrutamiento EM para agrupar cápsulas en una cápsula principal en tiempo de ejecución, es decir, si las cápsulas de nariz, boca y ojos tienen un voto con un valor de matriz de pose similar, se agrupan para formar una cápsula principal: la cápsula de la cara.

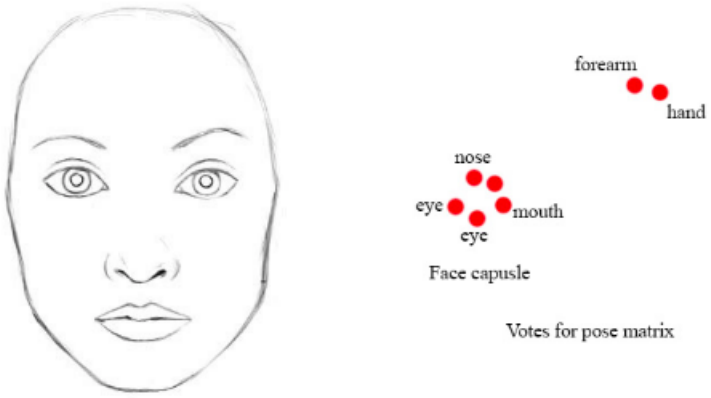

Figura 2.14: Cápsulas de un rostro.
Fuente: (Hui, 2017)

Un voto v_{ij} para la cápsula principal j de la cápsula i se calcula multiplicando la matriz de pose M_i de la cápsula i con una matriz de transformación W_{ij}.

$$v_{ij} = M_i W_{ij} \tag{2.16}$$

La probabilidad de que una cápsula i se agrupe en la cápsula j como una relación parte-todo se basa en la proximidad del voto v_{ij} a otros votos $(v_{o1j}...v_{okj})$ de otras cápsulas. W_{ij} se aprende de manera discriminatoria a través de una función de costo y la propagación hacia atrás. W_{ij} aprende no solo de qué está compuesta un objeto, sino también se asegura de que la información de pose de la cápsula principal coincida con sus cápsulas subcomponentes después de alguna transformación.

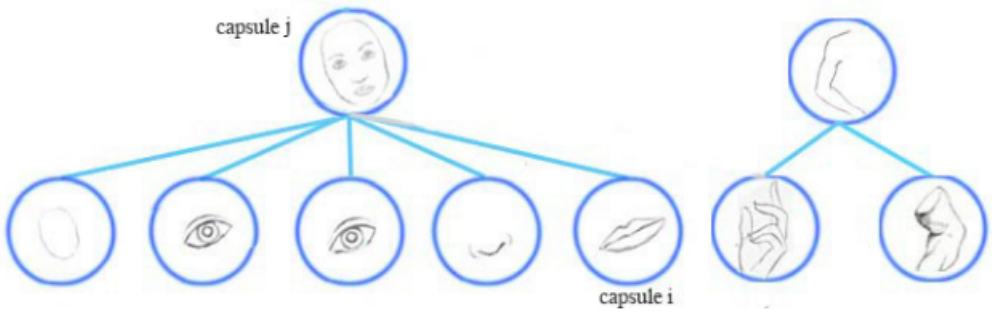

Figura 2.15: Agrupamiento de las cápsulas de rostro y brazo.
Fuente: (Hui, 2017) que en la figura viene a ser representado por

En la Figura 2.16. se puede apreciar el enrutamiento de las cápsulas matriciales, donde una cápsula representa la boca y la otra la nariz; ambos buscan agruparse o ser parte de un grupo mayor que viene a ser la cara, para ello las cápsulas con votos similares $(T_i T_{ij} = T_h T_{hj})$ se agrupan después de transformar sus matrices de pose (T_i y T_h) con una matriz de transformación W_{ij}, que en la figura viene a estar representado por T_{ij} y T_{hj} respectivamente. La activación de una cápsula está representado por P_i, P_h y P_j que vienen a ser la probabilidad de existencia de cada cápsula.

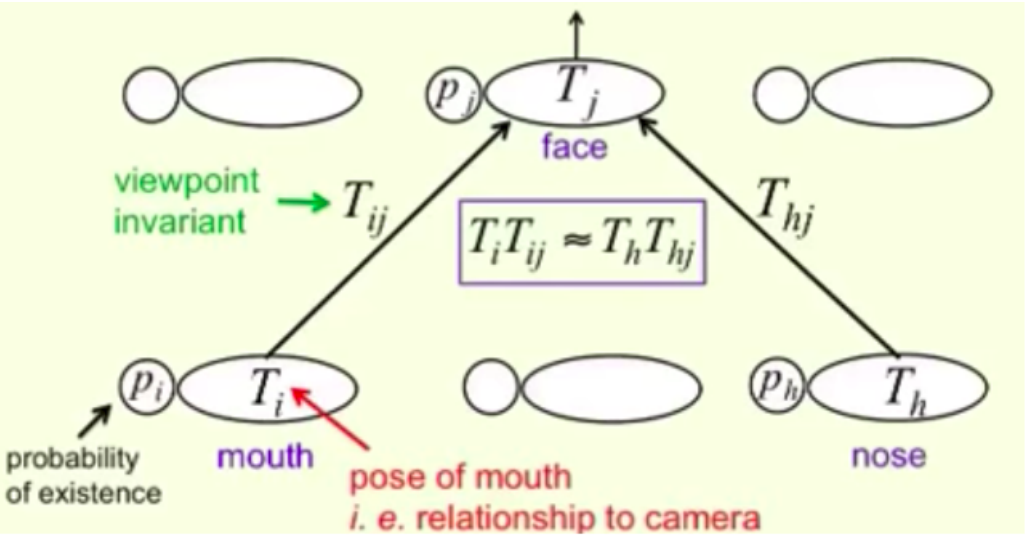

Figura 2.16: Esquema de enrutamiento EM de Cápsulas.
Fuente: (Hui, 2017)

Parte III

Desarrollo del proyecto

Capítulo 3

Modelos y Dataset

3.1. Modelos

3.1.1. DenseNet(Red Convolucional Densa)

DenseNet se compone de bloques densos. En estos bloques las capas están densamente conectadas entre sí, esto quiere decir que capa recibe en su entrada todos los mapas de características de salida de las capas anteriores.

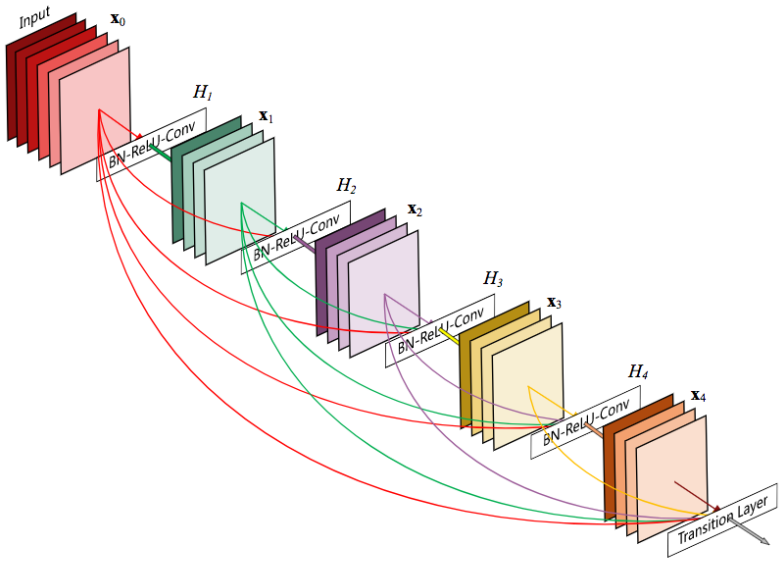

Figura 3.1: Modelo de DenseNet.
Fuente: https://www.kaggle.com/nepuerto/the-small-norb-dataset-v10

Este uso extremo de residuos crea una supervisión profunda porque cada capa recibe más supervisión de la función de pérdida gracias a las conexiones más cortas.

3.1.1.1. Bloques Densos(Dense Block)

Un bloque denso es un grupo de capas conectadas a todas sus capas anteriores, donde este compuesto por:

- Batch Normalization
- RELU
- Convolución 3x3

Los autores encontraron que el modo de preactivación (BN y RELU antes de la Conv) era más eficiente que el modo habitual de post-activación.

3.1.1.2. Tasa de Crecimiento(Growth Rate)

Concatenar los residuos en lugar de sumarlos tiene un inconveniente cuando el modelo es muy profundo "Genera muchos canales de entrada". La tasa de crecimiento regula la cantidad de información nueva que cada capa aporta al estado global. El número de mapas de características de salida de una capa se define como la tasa de crecimiento. Los autores recomiendan una tasa de crecimiento de 32 para un rendimiento óptimo, pero también se obtuvieron buenos resultados con una tasa de crecimiento de 12, por ende, en la implementación se realizar con tasas de crecimiento de 12 y 32.

3.1.1.3. Capas de Transición(Transition Layer)

En lugar de sumar el residuo como en ResNet, DenseNet concatena todos los mapas de características.Sería impracticable concatenar mapas de características de diferentes tamaños (aunque algunos cambios de tamaño pueden funcionar). Por lo tanto, en cada bloque denso, los mapas de características de cada capa tienen el mismo tamaño gracias a las capas de transición que están entre dos bloques densos. Una capa de transición está compuesta por:

- Batch Normalization
- 1x1 convolución (reduce la cantidad de canales en el volumen 3D, acelerando así el cálculo.)
- Pooling

3.1.1.4. Cuello de Botella(Bottleneck)

Una razón por la que DenseNet tiene pocos parámetros a pesar de concatenar muchos residuos juntos es que en cada convolución 3x3 se puede actualizar con un cuello de botella. Supongamos que con una tasa de crecimiento de 32, la última capa tendría de entrada 288 mapas de características. Gracias al cuello de botella esto se reduce, se tendría como máximo 128 mapas de características que alimentarían a una capa. Una capa de un bloque denso con un cuello de botella esta compuesta por:

- Batch Normalization
- RELU
- 1x1 cuello de botella de convolución que produce: tasa de crecimiento $*$ 4 mapas de características.
- Convolución 3x3

3.1.2. Cápsula de matriz con Enrutamiento EM

La arquitectura consta de una capa convolucional, seguida de dos capas de cápsulas, y una capa de cápsulas de clasificación final.

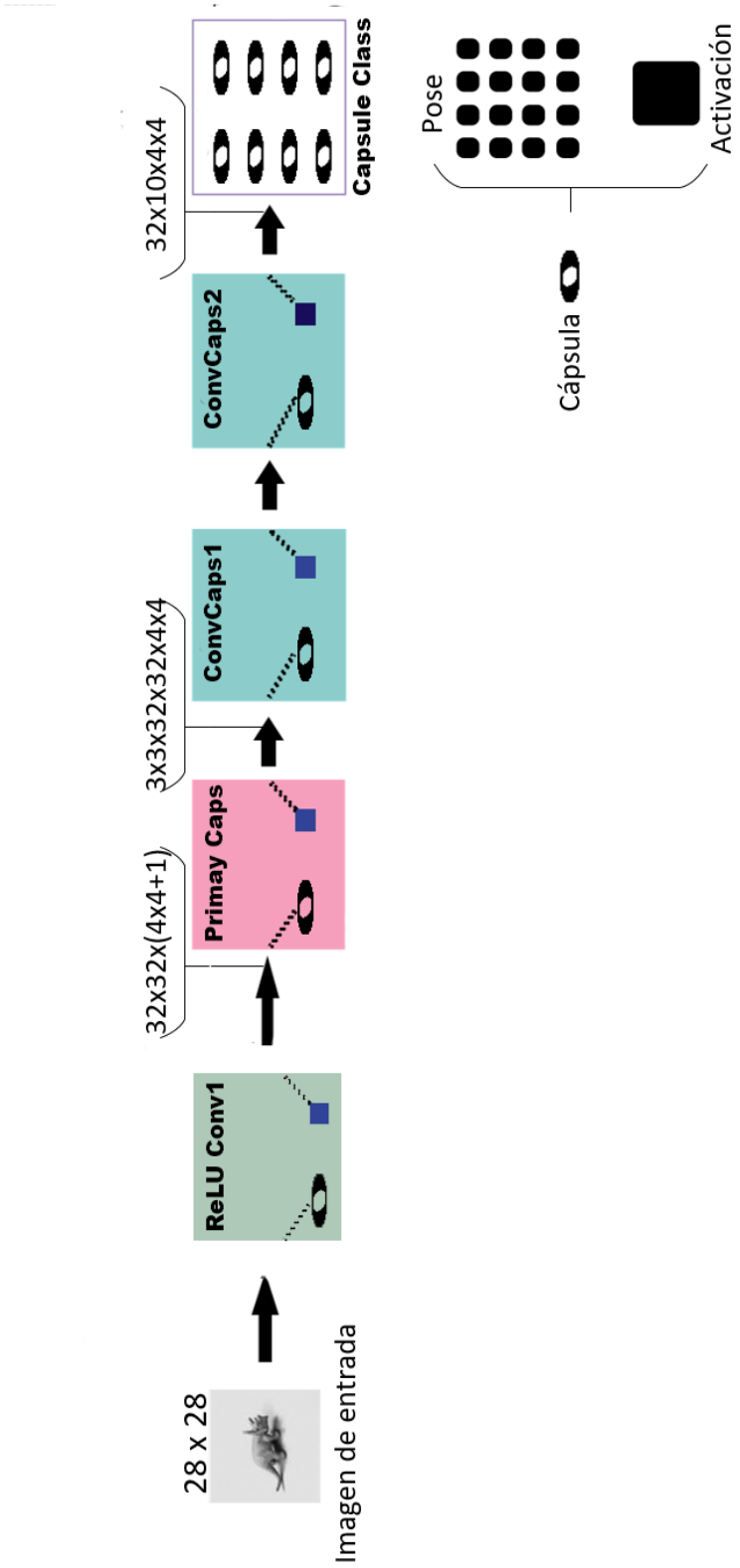

Figura 3.2: Modelo Base.
Fuente: Creación propia

RELU Conv1 es una capa de convolución simple (CNN) que utiliza un filtro 5×5 con un stride de 2 que genera 32 mapas de características.

En PrimaryCaps, se aplica un filtro de convolución 1×1 para transformar cada uno de los 32 canales en 32 cápsulas primarias. Cada cápsula contiene una matriz de pose 4×4 y un valor de activación. Se usa una capa de convolución simple para implementar PrimaryCaps. Se agrupa $4 \times 4 + 1$ neuronas para generar 1 cápsula.

PrimaryCaps es seguido por una capa de cápsula convolucional ConvCaps1 usando filtros 3×3 con un stride de 2. ConvCaps1 toma cápsulas como cápsulas de entrada y salida. ConvCaps1 es similar a una capa de convolución simple, excepto que utiliza el enrutamiento EM para calcular la salida de la cápsula.

La salida de la cápsula de ConvCaps1 alimenta a ConvCaps2. ConvCaps2 es otra capa de cápsula de convolución simple, pero con un stride de 1.

Las cápsulas de salida de ConvCaps2 están conectadas a una clase de capsulas CapsuleClass, utilizando un filtro 1x1 genera una cápsula por clase(En SmallNORB se tiene 10 clases).

3.2. Obtención del Dataset

Es necesario mencionar que el dataset utilizado para el entrenamiento es smallNORB, el cual presenta imágenes de juguetes en 3D, los cuales fueron tomadas en 6 condiciones de iluminación, 9 elevaciones (30 a 70 grados) y 18 azimuts (0 a 30 grados). El dataset smallNORB está dividido en 5 categorías: animales de cuatro patas, figuras humanas, aviones, camiones y automóviles.

El dataset smallNorb lo podemos descargar desde su página oficial THE small NORB DATASET, V1.0[1].Los archivos están en un formato de matriz binaria simple. Los archivos "-dat" almacenan las secuencias de imágenes. Los archivos "–cat" almacenan la categoría correspondiente de las imágenes. Cada archivo "–dat" almacena 24.300 pares de imágenes (5 categorías, 5 instancias, 6 iluminaciones, 9 elevaciones y 18 azimuths). El archivo "–cat" contiene 24.300 etiquetas de categoría (0 para animales, 1 para humanos, 2 para avión, 3 para camión, 4 para automóvil). Cada archivo "–info" almacena 24.300 vectores de 4 dimensiones, que contienen información adicional sobre las imágenes correspondientes:

[1]Página: https://cs.nyu.edu/ ylclab/data/norb-v1.0-small/

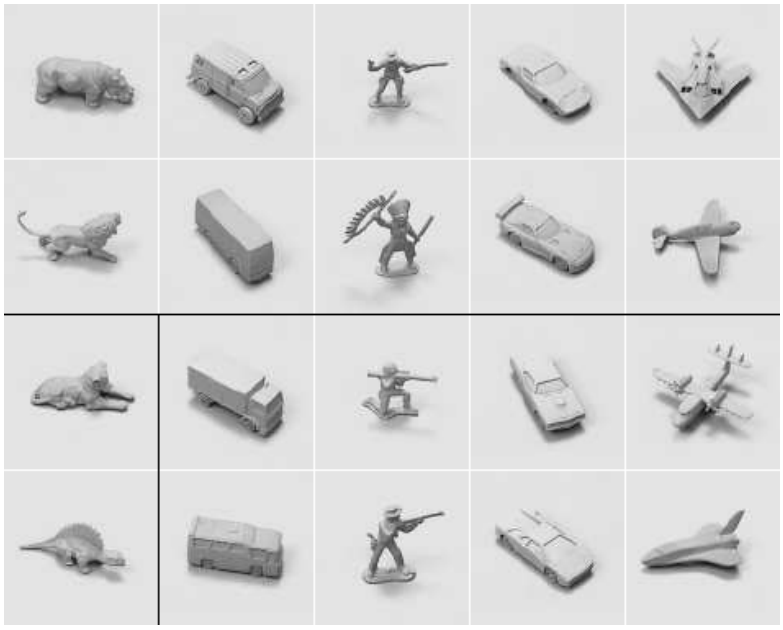

Figura 3.3: Imágenes del dataset SmallNorb.
Fuente: https://www.kaggle.com/nepuerto/the-small-norb-dataset-v10

3.3. Diseño del Modelo

Como ya se vio, el modelo base, Matriz de cápsula con enrutamiento EM, está compuesto por 5 capas: ReLU Conv1, Primary Caps, ConvCaps1, ConvsCaps2 y Capsule Class. Por ello se propone que para el nuevo modelo reemplazar la primera capa (ReLU Conv1) por una red convolucional densa Figura 3.4, presentado así la siguiente nueva estructura de capas:

- DenseNet
- Primary Caps.
- ConvCaps1.
- ConvCaps2.
- Capsule Class

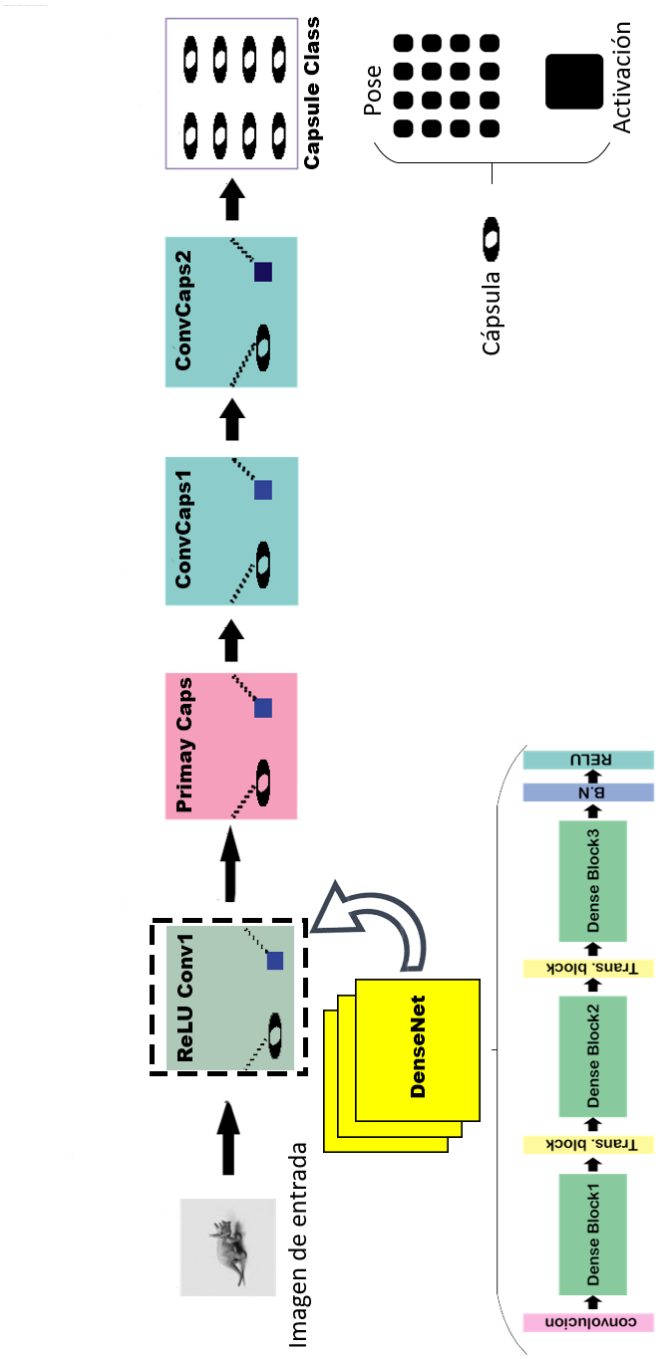

Figura 3.4: Reemplazo de la primera capa ReLU Conv1 por la capa Densenet.
Fuente: Creación propia

Para el diseño del modelo propuesto, esté constará de dos subredes:

- DenseNet.
- EM Routing.

Este modelo es desarrollado a través de un proceso experimental. A continuación, se describe cada subred del modelo propuesto.

3.3.1. Densenet

La red Convolucional Densa estará compuesta de 3 bloques densos, 2 bloques de transición, una función de Batch Normalization (BN) y una función de activación RELU. Un bloque denso comprende n(n=10) capas densas. Estas capas densas se conectan mediante un circuito denso de manera que cada capa densa recibe mapas de características de todas las capas anteriores y pasa sus mapas de características a todas las capas posteriores. La tasa de crecimiento (Growth rate) será de 12 y 32, ya que para hacer experimentos son valores que más se utiliza. Se utilizará un cuello de botella para reducir el excedente de mapas de características. De toda esta red nos dará una salida de 32 canales o mapas de características que se conectan con la capa PrimaryCaps.

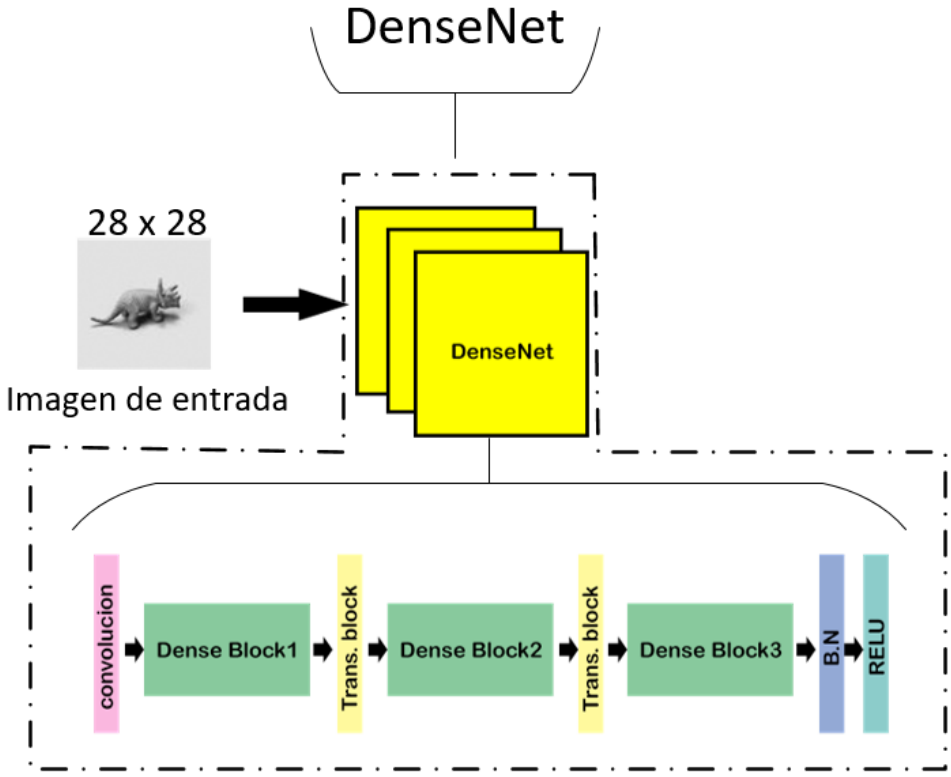

Figura 3.5: Sub Red DenseNet.
Fuente: Creación propia

3.3.2. EM Routing

PrimaryCaps Las cápsulas primarias se generan modificando los canales o mapas de características de salida de la subred DenseNet. En la capa PrimaryCaps, se aplica un filtro de convolución 1x1 para transformar cada uno de los 32 mapas de características en 32 capsulas primarias. Cada capsula contiene una matriz de pose 4x4 (para obtener las posición y orientación relativa de cada objeto). Por ende, se usa una capa de convolución regular para implementar capsulas primarias, por ende, se agrupa 32 x (4x4 + 1) neuronas para generar 32 capsulas primarias.

ConvCaps1 y ConvCaps2 A PrimaryCaps le sigue una capa de cápsula de convolución ConvCaps1 que utiliza filtros 3x3 con un stride de 2. ConvCaps1 toma cápsulas de entrada y salida. ConvCaps1 es similar a una capa de convolución normal, excepto que utiliza el enrutamiento EM para calcular la salida de la cápsula. La salida de la cápsula de ConvCaps1 alimenta a ConvCaps2. ConvCaps2 es otra capa similar a ConvCaps1, pero con un stride de 1. Las cápsulas de salida de ConvCaps2 se conectan a Capsules-Class mediante un filtro 1x1 y el cual genera una capsula por clase. La salida de ConCaps2 tiene una forma de (24, 4 x 4, 32, 4 x 4), donde: 24 es el tamaño de lote, 4 x 4 es la salida espacial, 32 son los canales de salida, 4 x 4 es la matriz de pose.

Enrutamiento Em La matriz de pose y la activación de las cápsulas de salida se calculan iterativamente utilizando el enrutamiento EM. El método EM ajusta los datapoints en una mezcla de modelos gaussianos con llamadas a las funciones M-STEP y E-STEP, donde M-STEP calculara la media y la varianza de las cápsulas, E-STEP calcula el enrutamiento de una cápsula.

Capsules Class La salida de ConvCaps2 alimenta a la capa Capsule Class. En lugar de usar un filtro de 3x3 como en ConvCaps2, las Capsules Class usan un filtro de 1x1. Además, en lugar de generar una salida espacial 2D (6x6 en ConvCaps1 y 4x4 en ConvCaps2), genera 5 cápsulas, cada una representa una de las N clases en el SmallNORB. Capsule Class hace llamadas para calcular los votos y luego utilizar el enrutamiento EM para calcular las salidas de la cápsula. Para integrar la ubicación espacial de la clase predicha en la Capsule-Class, se suma las coordenadas escaladas del centro del campo receptivo de cada cápsula en ConvCaps2 a los dos primeros elementos de la votación, esto se define como adición de coordenadas. Según el documento Hinton y cols. (2018), esto alienta al modelo a entrenar a la matriz de transformación para producir valores para esos dos elementos con el fin de representar la posición de la característica en relación con el centro del campo receptivo de la cápsula. La motivación de la adición de coordenadas es tener los primeros 2 elementos de los votos para predecir las coordenadas espaciales de la clase predicha. Al retener la información espacial en la cápsula, se avanza más allá de simplemente verificar la presencia de una característica. Se alienta al sistema para verificar la relación espacial de las características y así evitar adversarios. es decir, si las ordenes espaciales de las características son incorrectas, entonces sus votos correspondientes no deberían coincidir.

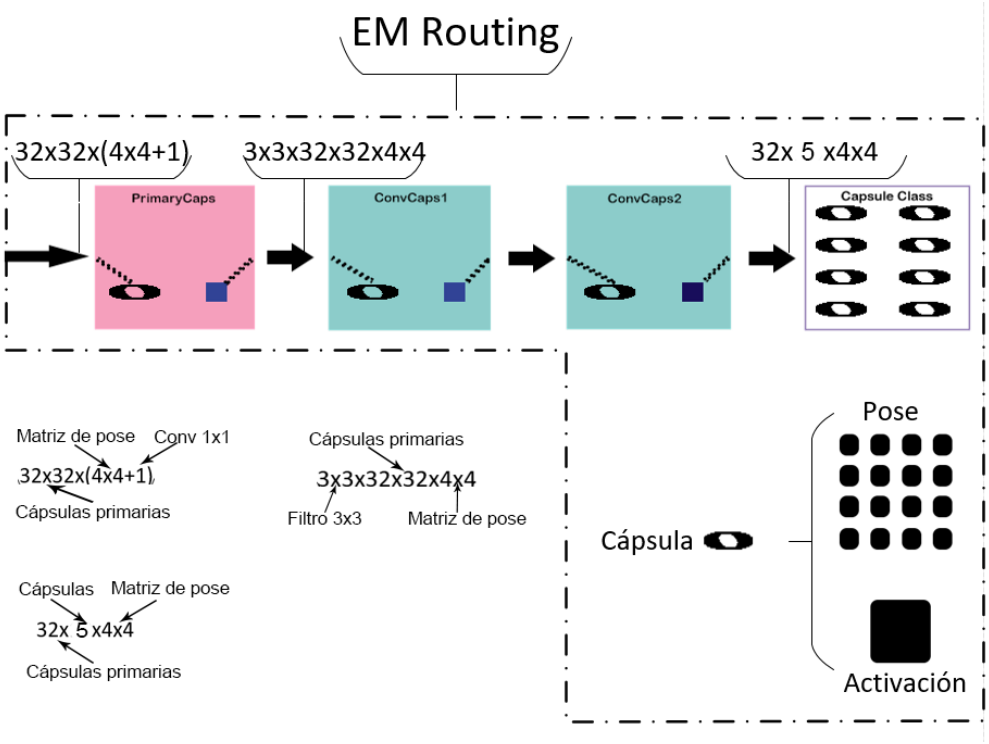

Figura 3.6: Subred EM Routing.
Fuente: Creación propia

Capa	Aplicación	Salida
SmallNorb		28,28,1
DenseNet	(1, 28, 28) − > 5x5 filtros, 32 canales de salida, stride =2,tasa de crecimiento=12 ó 32.	(32, 14, 14)
PrimaryCaps	(32, 14, 14) − > 1x1 filtro, 32 cápsulas, stride =1	Pose (14, 14, 32x4x4), Activación (14, 14, 32)
ConvCaps1	(14, 14, 32x (4x4 + 1)) − > 3x3 filtros, 32 cápsulas, stride = 2	Pose (6, 6, 32x4x4), Activación (6, 6, 32)
ConvCaps2	(6, 6, 32x (4x4 + 1)) − > 3x3 filtros, 32 cápsulas, stride = 1	Pose (4, 4, 32x4x4), Activación (4, 4, 32)
Class Capsules	(4, 4, 32x (4x4 + 1)) − > 1x1 conv, 5 cápsulas	Pose (5x4x4), Activación (5)

Tabla 3.1: Resumen de entradas y salidas de cada capa del modelo propuesto.
Fuente: Creación propia

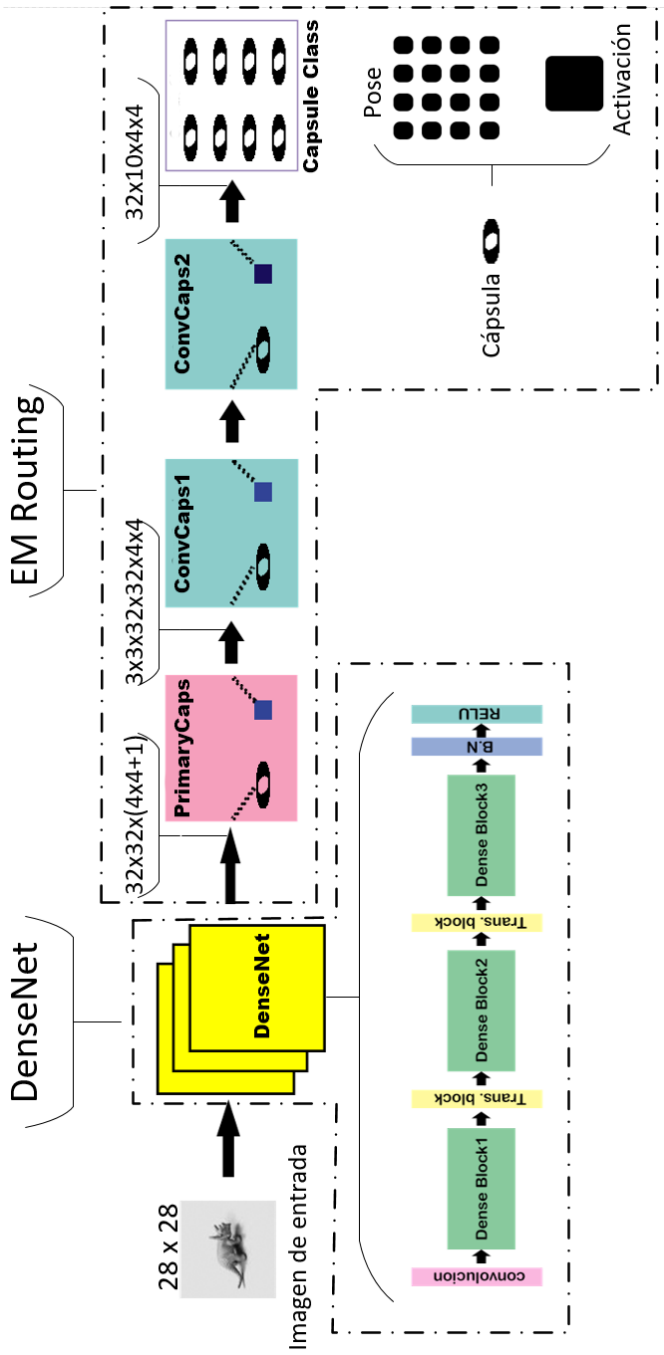

Figura 3.7: Modelo Propuesto.
Fuente: Creación propia

La Figura 3.7 representa el diseño final del modelo propuesto, el cual esta dividido en dos sub redes: Densenet y EM Routing. En la primera subred esta la capa DenseNet y en la según subred estarán las capas PrimaryCaps, ConvCaps1, ConvCaps2 y CapsuleClass.

3.4. Implementación del Modelo

3.4.1. Modelo Propuesto

Para la construcción del modelo propuesto ya se menciono que esté constará de dos subredes:

- DenseNet.
- EM Routing.

3.4.1.1. Densenet

Esta subred como se sabe mejora aún más el flujo de información entre capas, introduciendo conexiones directas de las capas anteriores hacia las capas posteriores.. Para la l-ésima capa se utiliza como entrada la siguiente función de composición Xl=Hl([X0, X1, . . . , Xl-1]) donde Hl(.) concatena las entradas múltiples [X0, X1, . . . , Xl-1] en un solo tensor. La función de composición Hl(.) consta de tres operaciones consecutivas: Batch Normalization (BN), seguida por Rectified Linear Unit (RELU) y una Convolución de 3x3 (Conv). Puesto que la función de concatenación no es viable cuando el tamaño de los mapas de características entrantes son diferentes, se utiliza las capas de transición que cambian el tamaño de los mapas de características. La subred se divide en tres bloques densos, cada uno con un número igual de capas. Las capas de transición entre dos bloques densos contiguos utilizadas en la subred, consisten en una capa de Batch Normalization, una capa de convolución de 1x1 seguida de una capa average pooling de 2x2. Las dimensiones de los mapas de características en los tres bloques densos son 32x32, 16x16 y 8x8, respectivamente. Al final del último bloque denso se realiza un average pooling global, obteniendo como 32 canales de salida.

De la Figura 3.8 se observa un segmento de código de la función BasicBlock que representa la implementación de un bloque básico, compuesta de un Batch Normalization (BN), Rectified Linear Unit (RELU) y una Convolución de 3x3 (Conv).

```
class BasicBlock(nn.Module):
    # Bloque basico
    def __init__(self, in_planes, out_planes, dropRate=0.0):
        super(BasicBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.droprate = dropRate
    def forward(self, x):
        out = self.conv1(self.relu(self.bn1(x)))
        if self.droprate > 0:
            out = F.dropout(out, p=self.droprate, training=self.training)
        return torch.cat([x, out], 1)
```

Figura 3.8: Código en Python de la función de un bloque basico (BasicBlock).
Fuente: Creación propia

De la Figura 3.9 se observa el código de la función DenseBlock en el cual nos genera un bloque denso a partir de un bloque básico (BasicBlock) y de las capas y tasa de crecimiento que se le asigne.

```python
class DenseBlock(nn.Module):
    # Bloque Denso
    def __init__(self, nb_layers, in_planes, tasa_crecimiento, block, dropRate=0.0):
        super(DenseBlock, self).__init__()
        self.layer = self._make_layer(block, in_planes, tasa_crecimiento, nb_layers, dropRate)
    def _make_layer(self, block, in_planes, tasa_crecimiento, nb_layers, dropRate):
        layers = []
        for i in range(nb_layers):
            layers.append(block(in_planes+i*tasa_crecimiento, tasa_crecimiento, dropRate))
        return nn.Sequential(*layers)
    def forward(self, x):
        return self.layer(x)
```

Figura 3.9: Código en Python de la función del bloque denso (DenseBlock).
Fuente: Creación propia

De la Figura 3.10 se observa el código de la función TransitionBlock que representa la implementación de una capa de transición, compuesta de un Batch Normalization (BN), Rectified Linear Unit (RELU) y una Convolución de 1x1 (Conv).

```python
class TransitionBlock(nn.Module):
    # Bloque de transición
    def __init__(self, in_planes, out_planes, dropRate=0.0):
        super(TransitionBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=1,
                               padding=0, bias=False)
        self.droprate = dropRate
    def forward(self, x):
        out = self.conv1(self.relu(self.bn1(x)))
        if self.droprate > 0:
            out = F.dropout(out, p=self.droprate, inplace=False, training=self.training)
        return F.avg_pool2d(out, 2)
```

Figura 3.10: Código en Python de las función del bloque de transición (TransitionBlock).
Fuente: Creación propia

3.4.1.2. EM Routing

PrimaryCaps Puesto que la primera subred tuvo 32 mapas de características de salida, entonces se generan 32 cápsulas primarias. Para generar 32 cápsulas primarias se aplica una de convolución de 1x1 con un stride de 1, luego se remodelan los canales obtenidos tomando 16 valores escalares para formar matrices de pose de 4x4 y finalmente se generan 32 valores de activación para las 32 cápsulas primarias. Lo anterior implica que cada cápsula primaria contará con una matriz de pose de 4x4 mas un valor de activación.

De la Figura 3.11 se observa un segmento de código de la función PrimaryCaps que representa la implementación de la capa de PrimaryCaps, compuesta de una

convolución de 1x1, una función de activación sigmoide y una matriz de pose.

```python
class PrimaryCaps(nn.Module):
    # capa primarycaps
    def __init__(self, A=32, B=32, K=1, P=4, stride=1):
        super(PrimaryCaps, self).__init__()
        self.pose = nn.Conv2d(in_channels=A, out_channels=B*P*P,
                              kernel_size=K, stride=stride, bias=True)
        self.a = nn.Conv2d(in_channels=A, out_channels=B,
                           kernel_size=K, stride=stride, bias=True)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        p = self.pose(x)
        a = self.a(x)
        a = self.sigmoid(a)
        out = torch.cat([p, a], dim=1)
        out = out.permute(0, 2, 3, 1)
        return out
```

Figura 3.11: Código en Python de la función del PrimaryCaps.
Fuente: Creación propia

ConvCaps1 y ConvCaps2 AComo se sabe a A PrimaryCaps le sigue una capa de capsula de convolución ConvCaps1 que usa 3x3 filtros con un stride de 2. Esta capa es similar a una capa de convolución normal, excepto que utiliza el enrutamiento EM para calcular la salida de la capsula. La salida de la capsula de ConvCaps1 alimenta a ConvCaps2 que es otra capa de capsula de convolución, pero con un stride de 1.

Codificacion del Enrutamiento Em El propósito principal del enrutamiento EM es calcular las matrices de pose y las activaciones de las cápsulas de salida. Se realizan N iteraciones, en la última iteración, el m_step completa el último cálculo de esos parámetros. Por lo tanto, se omite el e_step en la última iteración que es principalmente responsable de volver a calcular la asignación de enrutamiento.

```
def caps_em_routing(self, v, a_in, C, eps):

    b, B, c, psize = v.shape
    assert c == C
    assert (b, B, 1) == a_in.shape

    r = torch.cuda.FloatTensor(b, B, C).fill_(1./C)
    for iter_ in range(self.iters):
        a_out, mu, sigma_sq = self.m_step(a_in, r, v, eps, b, B, C, psize)
        if iter_ < self.iters - 1:
            r = self.e_step(mu, sigma_sq, a_out, v, eps, b, C)

    return mu, a_out
```

Figura 3.12: Código en Python de la función Em Routing.
Fuente: Creación propia

M-Steps Es la función que calcula la media y varianza de las capsulas para luego calcular la activación de estas. Para calcular la activación de una capsula, se usa la siguiente ecuación:

$$a_j = sigmoid(\lambda(b_j - \sum_h cost_j^h)) \tag{3.1}$$

Donde "b_j"se define como el costo de describir la media y la varianza de la cápsula j. λ es el parámetro de temperatura inversa $\frac{1}{temperature}$.En la implementación se comienza desde 1 y se incrementa en 1 después de cada iteración.$cost_j^h$ viene a ser el costo de activción de la capsula j.

```
def m_step(self, a_in, r, v, eps, b, B, C, psize):

    r = r * a_in
    r = r / (r.sum(dim=2, keepdim=True) + eps)
    r_sum = r.sum(dim=1, keepdim=True)
    coeff = r / (r_sum + eps)
    coeff = coeff.view(b, B, C, 1)

    mu = torch.sum(coeff * v, dim=1, keepdim=True)
    sigma_sq = torch.sum(coeff * (v - mu)**2, dim=1, keepdim=True) + eps

    r_sum = r_sum.view(b, C, 1)
    sigma_sq = sigma_sq.view(b, C, psize)
    cost_h = (self.beta_u.view(C, 1) + torch.log(sigma_sq.sqrt())) * r_sum

    a_out = self.sigmoid(self._lambda*(self.beta_a - cost_h.sum(dim=2)))
    sigma_sq = sigma_sq.view(b, 1, C, psize)

    return a_out, mu, sigma_sq
```

Figura 3.13: Código en Python de la función M-Steps.
Fuente: Creación propia

E-Steps Es el principal responsable de volver a calcular la asignación de enrutamiento después de que m_step actualice la activación de las capsulas de salida y los modelos gaussianos con nuevos μ y σ.

```
def e_step(self, mu, sigma_sq, a_out, v, eps, b, C):

    ln_p_j_h = -1. * (v - mu)**2 / (2 * sigma_sq) \
               - torch.log(sigma_sq.sqrt()) \
               - 0.5*self.ln_2pi

    ln_ap = ln_p_j_h.sum(dim=3) + torch.log(a_out.view(b, 1, C))
    r = self.softmax(ln_ap)
    return r
```

Figura 3.14: Código en Python de la función E-Steps.
Fuente: Creación propia

Capsules Class CapsuleClass usa un filtro de 1x1. Además. Capsule Class hace llamadas para calcular los votos y luego utilizar el enrutamiento EM para calcular las salidas de la cápsula. Luego se generarán n capsulas por cada clase que presente SmallNORB, como SmallNORB presenta 5 clases entonces se generarán 5 capsulas.

De la figura 3.15 se observa un segmento de código de la función ConvCaps, esta función realizara el trabajo de las ConvCaps1, ConvCaps2 y CapsuleClass, según sea el valor de sus entradas. Además se definirá con cuantas iteraciones trabajara el modelo.

```python
class ConvCaps(nn.Module):
    #funcion en el cual trabajaran Concaps1,Concaps2 y CapsuleClass
    def __init__(self, B=32, C=32, K=3, P=4, stride=2, iters=3,
                 coor_add=False, w_shared=False):
        super(ConvCaps, self).__init__()
        self.B = B
        self.C = C
        self.K = K
        self.P = P
        self.psize = P*P
        self.stride = stride
        self.iters = iters
        self.coor_add = coor_add
        self.w_shared = w_shared
        # constantes
        self.eps = 1e-8
        self._lambda = 1e-03
        self.ln_2pi = torch.cuda.FloatTensor(1).fill_(math.log(2*math.pi))
        # parametros
        self.beta_u = nn.Parameter(torch.zeros(C))
        self.beta_a = nn.Parameter(torch.zeros(C))
        self.weights = nn.Parameter(torch.randn(1, K*K*B, C, P, P))
        # op
        self.sigmoid = nn.Sigmoid()
        self.softmax = nn.Softmax(dim=2)
```

Figura 3.15: Código en Python de la función ConvCaps.
Fuente: Creación propia

Spread Loss Las cápsulas de matriz requieren una función de pérdida para entrenarse. Se escoge Spread Loss como la principal función de pérdida. La ecuación de Spread Loss es:

$$L = \sum_{i \neq t} (max(0, m - (a_t - a_i)))^2 \tag{3.2}$$

Si el margen entre la etiqueta verdadera y la clase incorrecta es menor que m, se penaliza por el cuadrado de $m - (a_t - a_i)$. Inicialmente m comienza como 0.2 y aumenta linealmente en 0.1 después de cada entrenamiento de época. m dejará de crecer después de alcanzar el máximo 0.9. Comenzar en un margen inferior ayuda al entrenamiento a evitar demasiadas cápsulas muertas durante la fase inicial.

```python
class SpreadLoss(_Loss):

    def __init__(self, m_min=0.2, m_max=0.9, num_class=10):
        super(SpreadLoss, self).__init__()
        self.m_min = m_min
        self.m_max = m_max
        self.num_class = num_class

    def forward(self, x, target, r):
        b, E = x.shape
        assert E == self.num_class
        margin = self.m_min + (self.m_max - self.m_min)*r

        at = torch.cuda.FloatTensor(b).fill_(0)
        for i, lb in enumerate(target):
            at[i] = x[i][lb]
        at = at.view(b, 1).repeat(1, E)

        zeros = x.new_zeros(x.shape)
        loss = torch.max(margin - (at - x), zeros)
        loss = loss**2
        loss = loss.sum() / b - margin**2

        return loss
```

Figura 3.16: Código en Python de la función que calcula el Spread Loss.
Fuente: Creación propia

3.4.2. Detalles Técnicos

Para desarrollar el presente proyecto se utilizo el siguiente hardware y software

3.4.2.1. Hardware

- Laptop.- Asus GX531G.
- GPU.- NVIDEA 1070GX, GDDR5 a 8 GHz, .

3.4.2.2. Sofware

- Editor de Codigo.- Visual Studio Code 1.38.
- Lenguaje de Programación.- Python 3.7.
- Librerias
    - Pytorch 0.4.1:Es una librería de aprendizaje automático de código abierto, utilizada para aplicaciones como visión computacional y el procesamiento del lenguaje natural

- Matplolib 3.1.1:Es una librería para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.
- Numpy 1.16 Librería de funciones matemáticas de alto nivel

Parte IV

Proceso Experimental

Capítulo 4

Entrenamiento de los modelos

En esta sección se realiza el entrenamiento de los modelos Modelo Propuesto y el Modelo Base(Capsula de Matriz con Enrutamiento Em)

4.1. DataSet

Como ya se menciono, el dataset utilizado para el entrenamiento es Small-NORB, el cual presenta imágenes de juguetes en 3D, los cuales fueron tomadas en 6 condiciones de iluminación, 9 elevaciones (30 a 70 grados) y 18 azimuts (0 a 30 grados). El dataset SmallNORB está dividido en 5 categorías: animales de cuatro patas, figuras humanas, aviones, camiones y automóviles.

4.2. Metricas de Evaluacion

En la fase de entrenamiento para realizar la evaluacion del desenpeño de cada modelo se aplico las siguientes metricas:

- Precisión(Accuray).- Es la relación entre el número de predicciones correctas y el número total de muestras de entrada

$$Precisión(Accuray) = \frac{Numero - de - predicciones - correctas}{numero - total - de - entradas} \tag{4.1}$$

- Pérdida de Entrenamiento(Training loss) Para calcular la pérdida de entrenamiento se utilizará la ecuacion de Spread Loos ya viasta con anterioridad, cuya ecuación es:

$$L = \sum_{i \neq t} (max(0, m - (a_t - a_i)))^2 \tag{4.2}$$

- Tiempo.- Tiempo empleado en el entrenamiento

4.3. Entrenamiento de los modelos

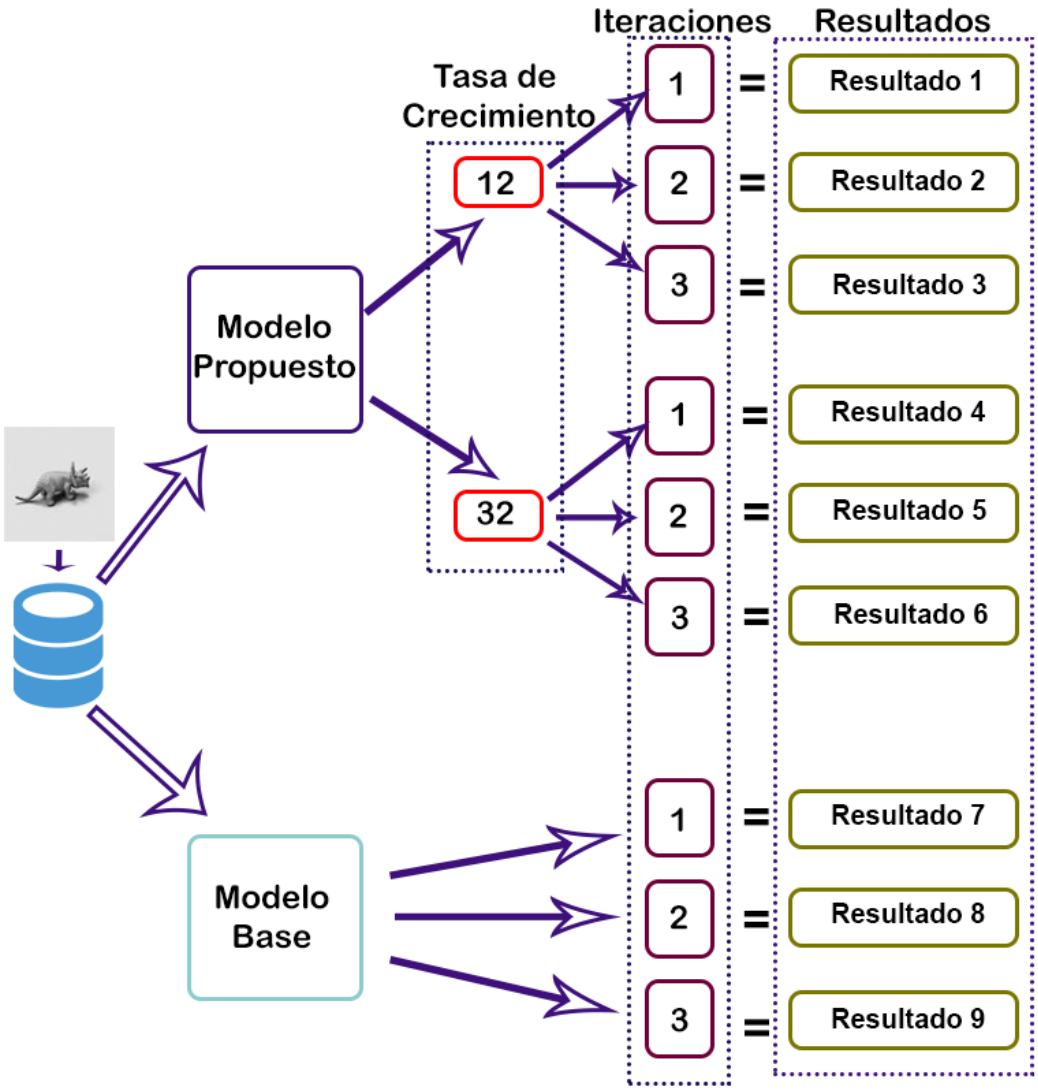

Figura 4.1: Esquema de experimentación.
Fuente: Creación propia

Para evaluar el rendimiento de los modelos, se medirá las métricas como son: Precisión, Pérdida y tiempo de entrenamiento, De la Figura 4.1 se observa que para obtener los resultados se plantea los siguientes casos.

- Medición de las metricas para el modelo propuesto con iteraciones de 1,2 y 3, con tazas de crecimiento de 12 y 32 respectivamente.
- Medición de las metricas para el modelo base con iteraciones de 1,2 y 3.

La medición se realizó con un entrenamiento de 30 epocas.

Epoca		Pérdida(training loss)	Precisión(Accuracy)	Tiempo
Epoca: 1	[36160/48600 (74%)]	training loos: 1.880435	Precision(Accuracy): 62.500000	Tiempo 0.189 (0.166)
Epoca: 1	[36480/48600 (75%)]	training loos: 1.950686	Precision(Accuracy): 53.125000	Tiempo 0.187 (0.167)
Epoca: 1	[36800/48600 (76%)]	training loos: 1.981167	Precision(Accuracy): 37.500000	Tiempo 0.187 (0.167)
Epoca: 1	[37120/48600 (76%)]	training loos: 2.071859	Precision(Accuracy): 37.500000	Tiempo 0.141 (0.167)
Epoca: 1	[37440/48600 (77%)]	training loos: 2.021399	Precision(Accuracy): 50.000000	Tiempo 0.141 (0.167)
Epoca: 1	[37760/48600 (78%)]	training loos: 2.037937	Precision(Accuracy): 56.250000	Tiempo 0.141 (0.166)
Epoca: 1	[38080/48600 (78%)]	training loos: 2.114617	Precision(Accuracy): 53.125000	Tiempo 0.141 (0.166)
Epoca: 1	[38400/48600 (79%)]	training loos: 2.114870	Precision(Accuracy): 40.625000	Tiempo 0.135 (0.166)
Epoca: 1	[38720/48600 (80%)]	training loos: 2.085577	Precision(Accuracy): 59.375000	Tiempo 0.125 (0.166)
Epoca: 1	[39040/48600 (80%)]	training loos: 2.177144	Precision(Accuracy): 37.500000	Tiempo 0.125 (0.165)
Epoca: 1	[39360/48600 (81%)]	training loos: 2.232128	Precision(Accuracy): 43.750000	Tiempo 0.135 (0.165)
Epoca: 1	[39680/48600 (82%)]	training loos: 2.219991	Precision(Accuracy): 46.875000	Tiempo 0.141 (0.165)
Epoca: 1	[40000/48600 (82%)]	training loos: 2.267858	Precision(Accuracy): 40.625000	Tiempo 0.125 (0.165)
Epoca: 1	[40320/48600 (83%)]	training loos: 2.243769	Precision(Accuracy): 59.375000	Tiempo 0.125 (0.164)
Epoca: 1	[40640/48600 (84%)]	training loos: 2.325854	Precision(Accuracy): 43.750000	Tiempo 0.141 (0.164)
Epoca: 1	[40960/48600 (84%)]	training loos: 2.284602	Precision(Accuracy): 56.250000	Tiempo 0.125 (0.164)
Epoca: 1	[41280/48600 (85%)]	training loos: 2.326180	Precision(Accuracy): 46.875000	Tiempo 0.141 (0.164)
Epoca: 1	[41600/48600 (86%)]	training loos: 2.354798	Precision(Accuracy): 43.750000	Tiempo 0.136 (0.164)
Epoca: 1	[41920/48600 (86%)]	training loos: 2.375555	Precision(Accuracy): 56.250000	Tiempo 0.141 (0.163)
Epoca: 1	[42240/48600 (87%)]	training loos: 2.444918	Precision(Accuracy): 46.875000	Tiempo 0.125 (0.163)
Epoca: 1	[42560/48600 (88%)]	training loos: 2.480422	Precision(Accuracy): 40.625000	Tiempo 0.125 (0.163)
Epoca: 1	[42880/48600 (88%)]	training loos: 2.408301	Precision(Accuracy): 68.750000	Tiempo 0.141 (0.163)
Epoca: 1	[43200/48600 (89%)]	training loos: 2.543169	Precision(Accuracy): 43.750000	Tiempo 0.135 (0.163)
Epoca: 1	[43520/48600 (90%)]	training loos: 2.491839	Precision(Accuracy): 56.250000	Tiempo 0.141 (0.162)
Epoca: 1	[43840/48600 (90%)]	training loos: 2.537193	Precision(Accuracy): 43.750000	Tiempo 0.141 (0.162)
Epoca: 1	[44160/48600 (91%)]	training loos: 2.583767	Precision(Accuracy): 40.625000	Tiempo 0.134 (0.162)
Epoca: 1	[44480/48600 (92%)]	training loos: 2.640938	Precision(Accuracy): 50.000000	Tiempo 0.141 (0.162)
Epoca: 1	[44800/48600 (92%)]	training loos: 2.641425	Precision(Accuracy): 56.250000	Tiempo 0.154 (0.162)
Epoca: 1	[45120/48600 (93%)]	training loos: 2.667960	Precision(Accuracy): 37.500000	Tiempo 0.125 (0.161)
Epoca: 1	[45440/48600 (93%)]	training loos: 2.692296	Precision(Accuracy): 50.000000	Tiempo 0.141 (0.161)
Epoca: 1	[45760/48600 (94%)]	training loos: 2.685854	Precision(Accuracy): 50.000000	Tiempo 0.154 (0.161)
Epoca: 1	[46080/48600 (95%)]	training loos: 2.783611	Precision(Accuracy): 31.250000	Tiempo 0.141 (0.161)
Epoca: 1	[46400/48600 (95%)]	training loos: 2.835702	Precision(Accuracy): 50.000000	Tiempo 0.141 (0.161)
Epoca: 1	[46720/48600 (96%)]	training loos: 2.823829	Precision(Accuracy): 59.375000	Tiempo 0.125 (0.161)
Epoca: 1	[47040/48600 (97%)]	training loos: 2.826513	Precision(Accuracy): 46.875000	Tiempo 0.125 (0.160)
Epoca: 1	[47360/48600 (97%)]	training loos: 2.907299	Precision(Accuracy): 34.375000	Tiempo 0.125 (0.160)
Epoca: 1	[47680/48600 (98%)]	training loos: 2.916008	Precision(Accuracy): 53.125000	Tiempo 0.152 (0.160)
Epoca: 1	[48000/48600 (99%)]	training loos: 2.923934	Precision(Accuracy): 43.750000	Tiempo 0.134 (0.160)
Epoca: 1	[48320/48600 (99%)]	training loos: 3.022934	Precision(Accuracy): 50.000000	Tiempo 0.125 (0.160)

Figura 4.2: Proceso de entrenamiento vista desde la consola,
Fuente: Creación propia

4.3.1. Modelo Propuesto

4.3.1.1. Medición de las métricas con iteraciones de 1,2 y 3, con tasas de crecimiento de 12

la medición de las métricas utilizando iteraciones de 1,2 y 3 se detallan en las tablas 4.1, 4.2 y 4.3:

Iteracion de 1

De la Figura 4.3 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 67.72 %.

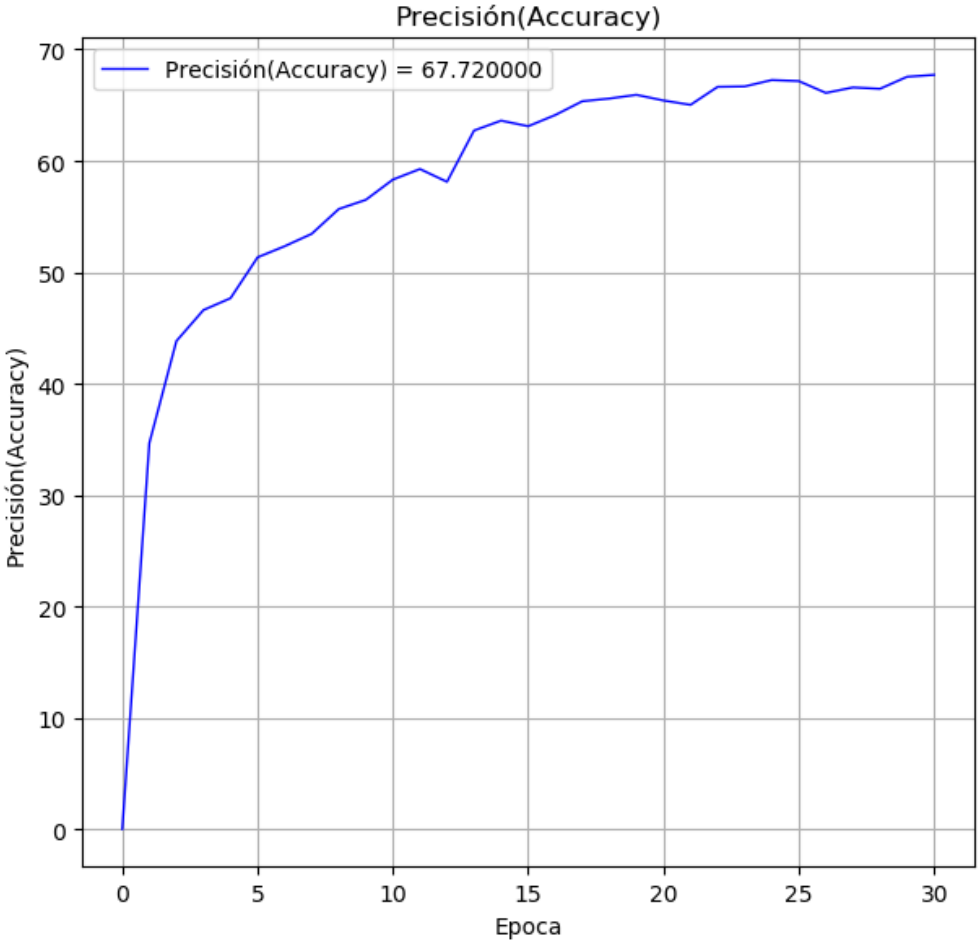

Figura 4.3: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.4 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 2.75 %.

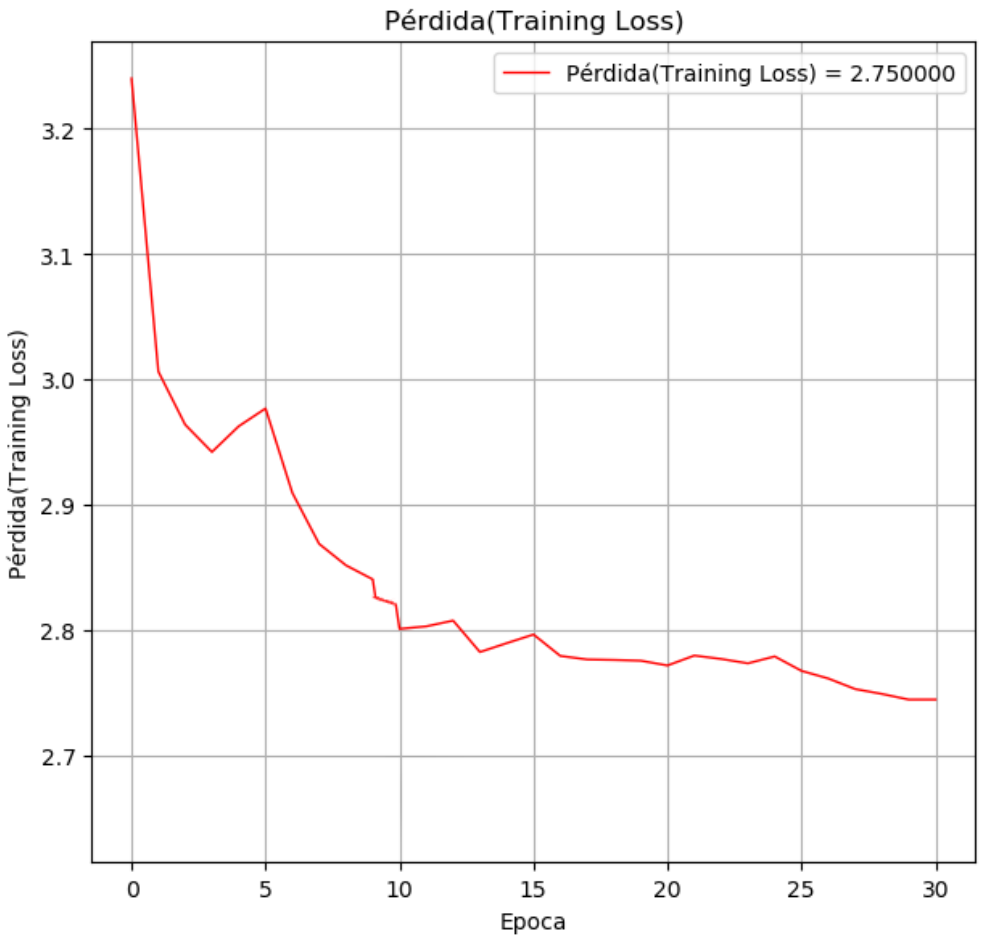

Figura 4.4: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.5 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 4 minutos, realizando un tiempo total de entrenamiento de 2 horas y 13 minutos.

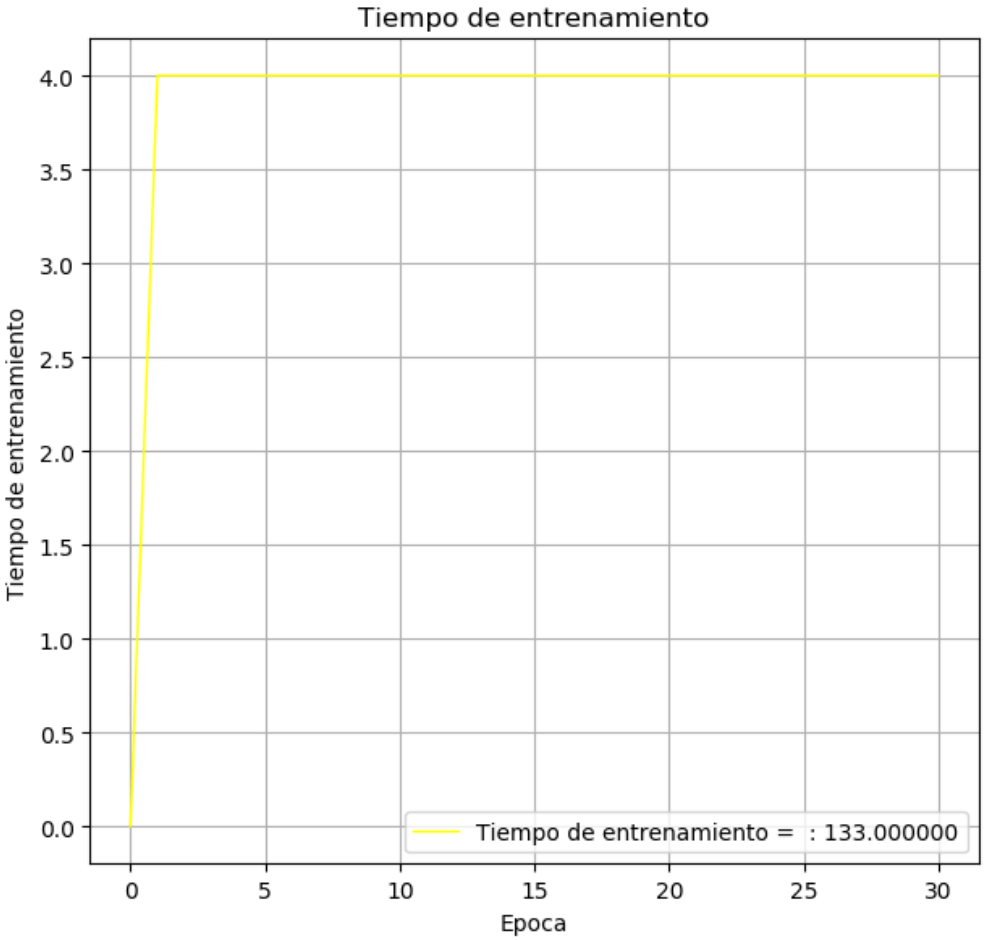

Figura 4.5: Medición del Tiempo en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Tabla 4.1 que corresponde a entrenamientos utilizando una iteración de 1, del cual observamos que el resultado final para la precisión(accuracy) es 67.72, en cuanto a la pérdida(training loss)es de 2.75 y el tiempo de entrenamiento fue de 2 horas, 13 minutos.

Iteración 1	Tasa de Crecimiento 12
Precisión(Accuracy)	67.72 %
Pérdida(Training Loss)	2.75 %
Tiempo	2 hora, 13 minutos

Tabla 4.1: Resultados para una iteracion de 1.

Fuente: Creación propia

Iteracion de 2

De la Figura 4.6 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 92.65 %.

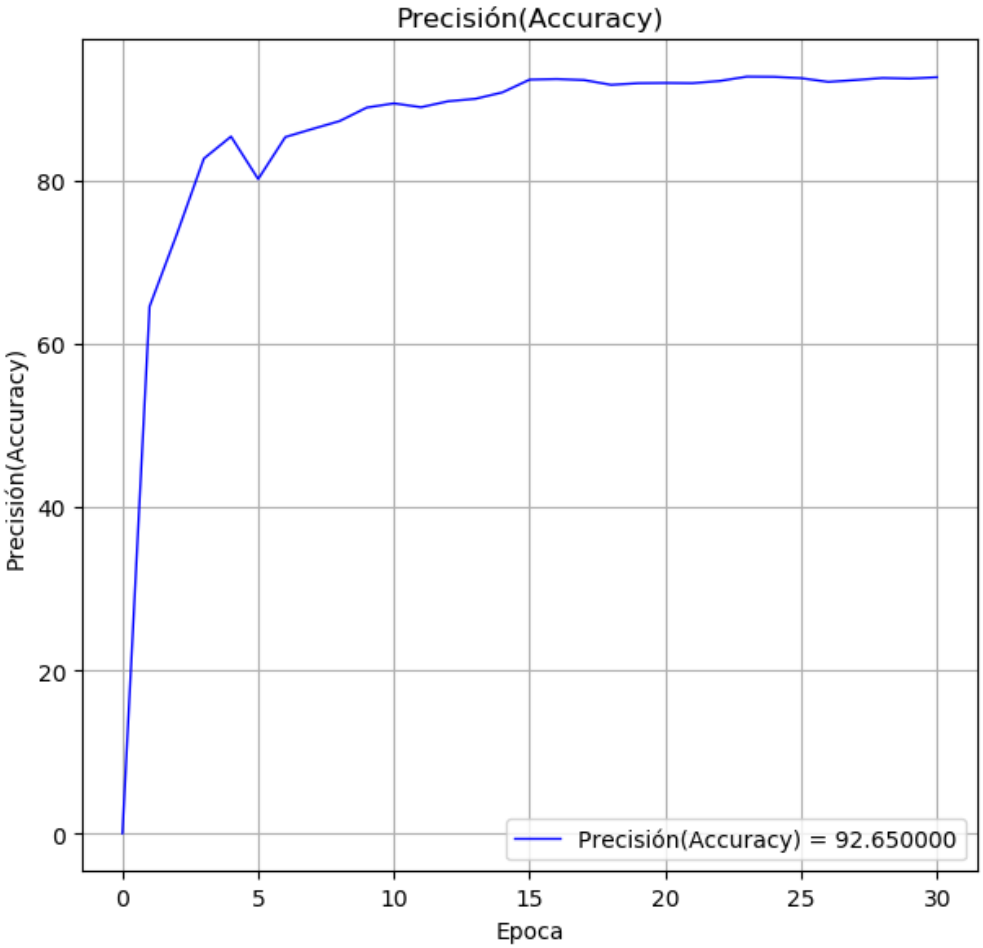

Figura 4.6: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.7 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 0.83 %.

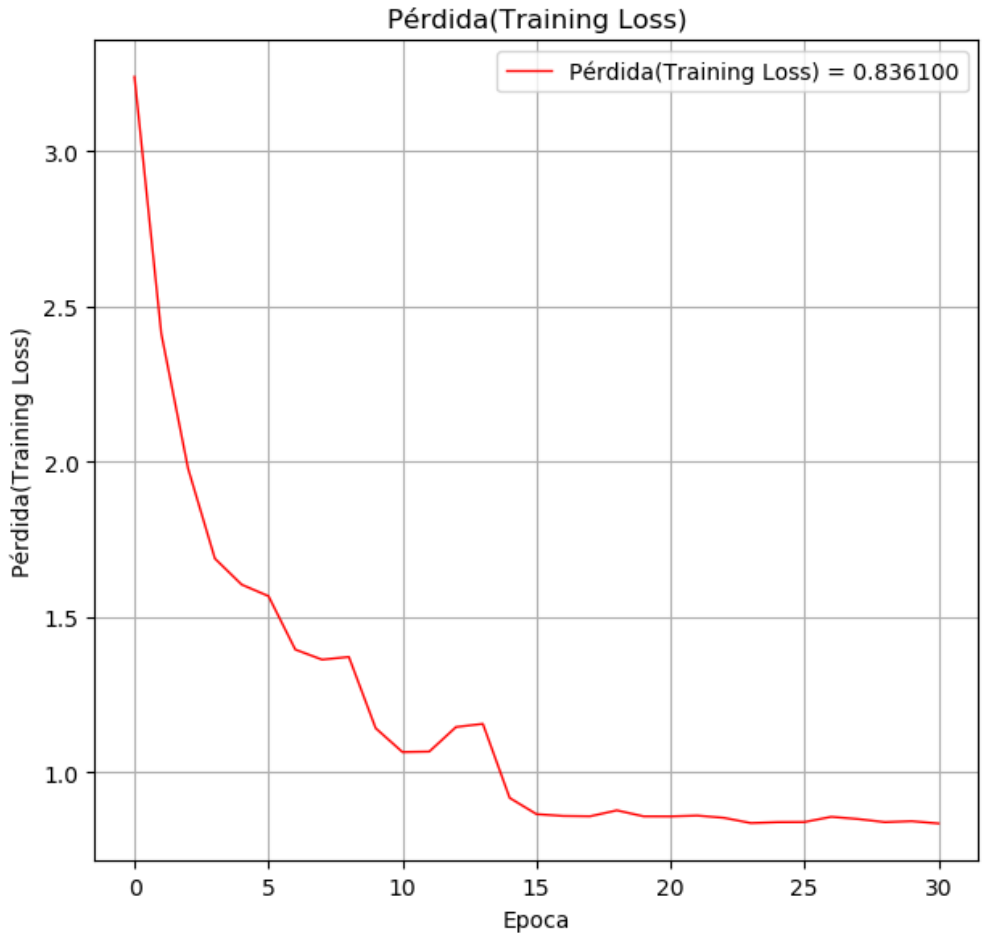

Figura 4.7: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.8 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 4 a 5 minutos, realizando un tiempo total de entrenamiento de 2 horas y 32 minutos.

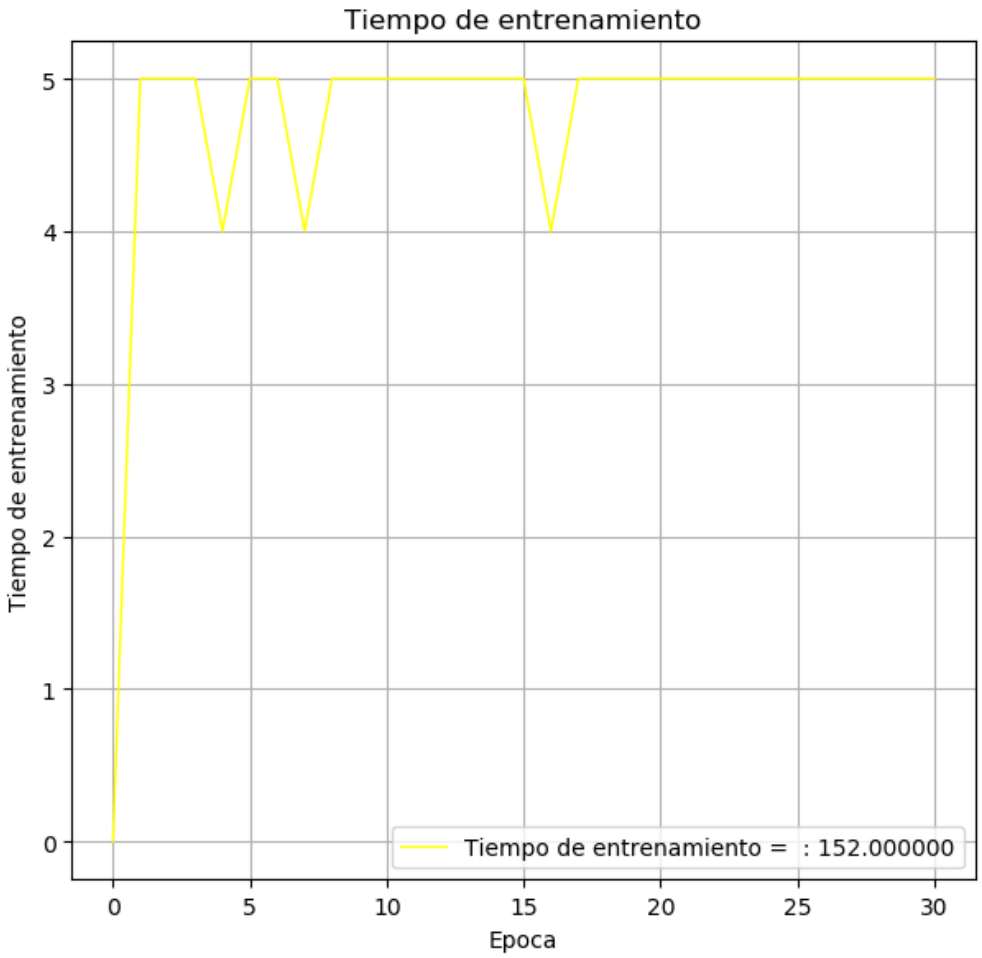

Figura 4.8: Medición del tiempo en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Tabla 4.2 que corresponde a entrenamientos utilizando una iteración de 2, del cual observamos que el resultado final para la precisión(accuracy) es 92.65, en cuanto a la pérdida(training loss)es de 0.83 y el tiempo de entrenamiento fue de 2 horas, 32 minutos.

Iteración 2	Tasa de Crecimiento 12
Precisión(Accuracy)	92.65 %
Pérdida(Training Loss)	0.83 %
Tiempo	2 horas, 32 minutos

Tabla 4.2: Resultados para una iteracion de 2.

Fuente: Creación propia

Iteracion de 3

De la Figura 4.9 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 32.77 %.

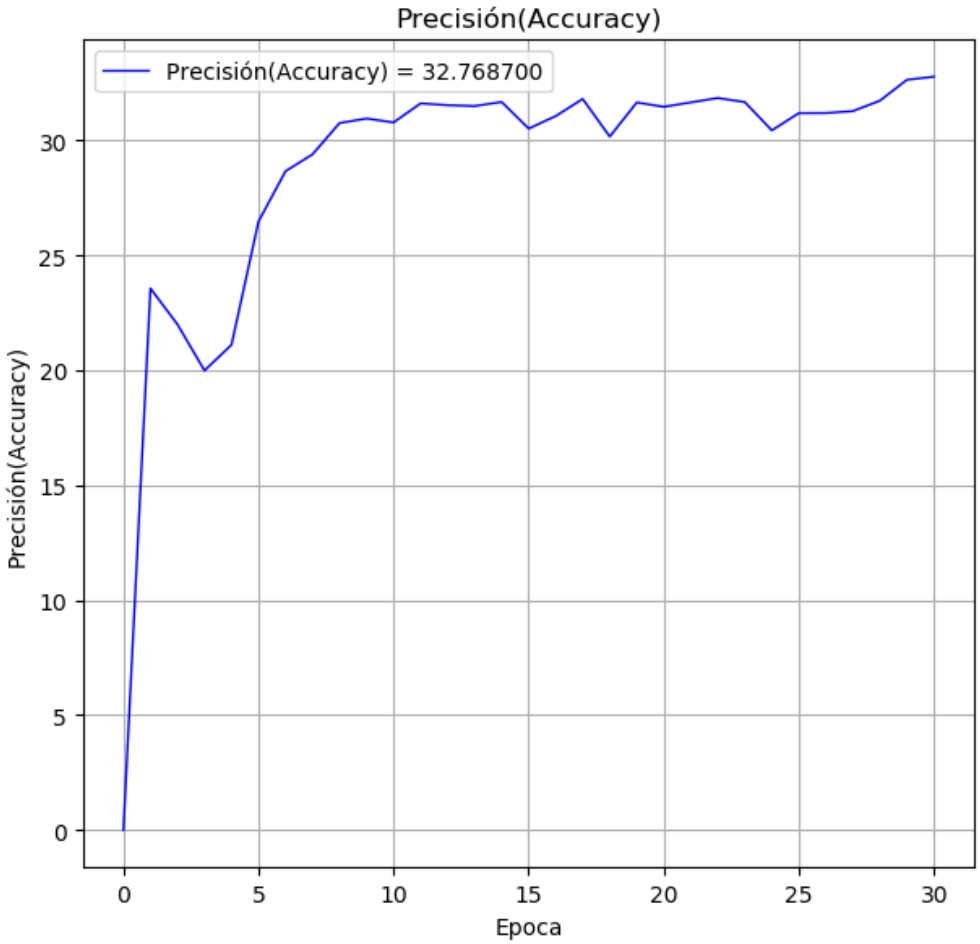

Figura 4.9: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.10 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 2.81 %.

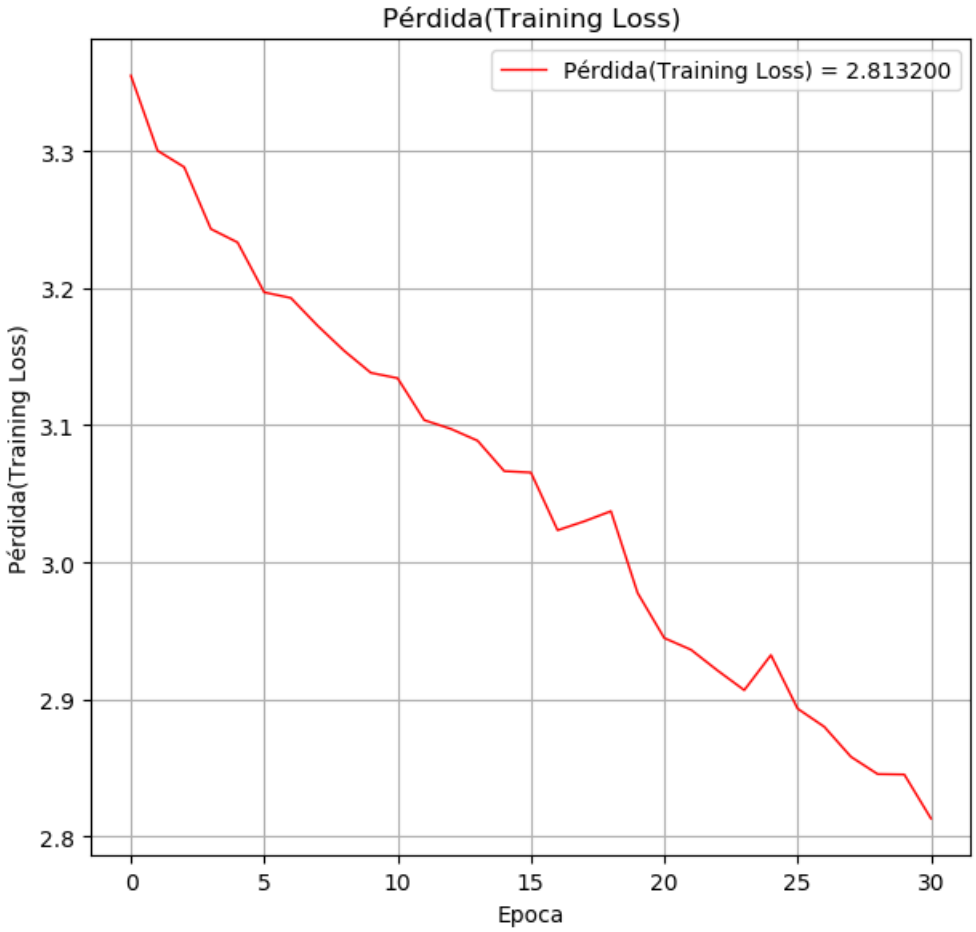

Figura 4.10: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Figura 4.11 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 5 minutos, realizando un tiempo total de entrenamiento de 2 horas y 57 minutos.

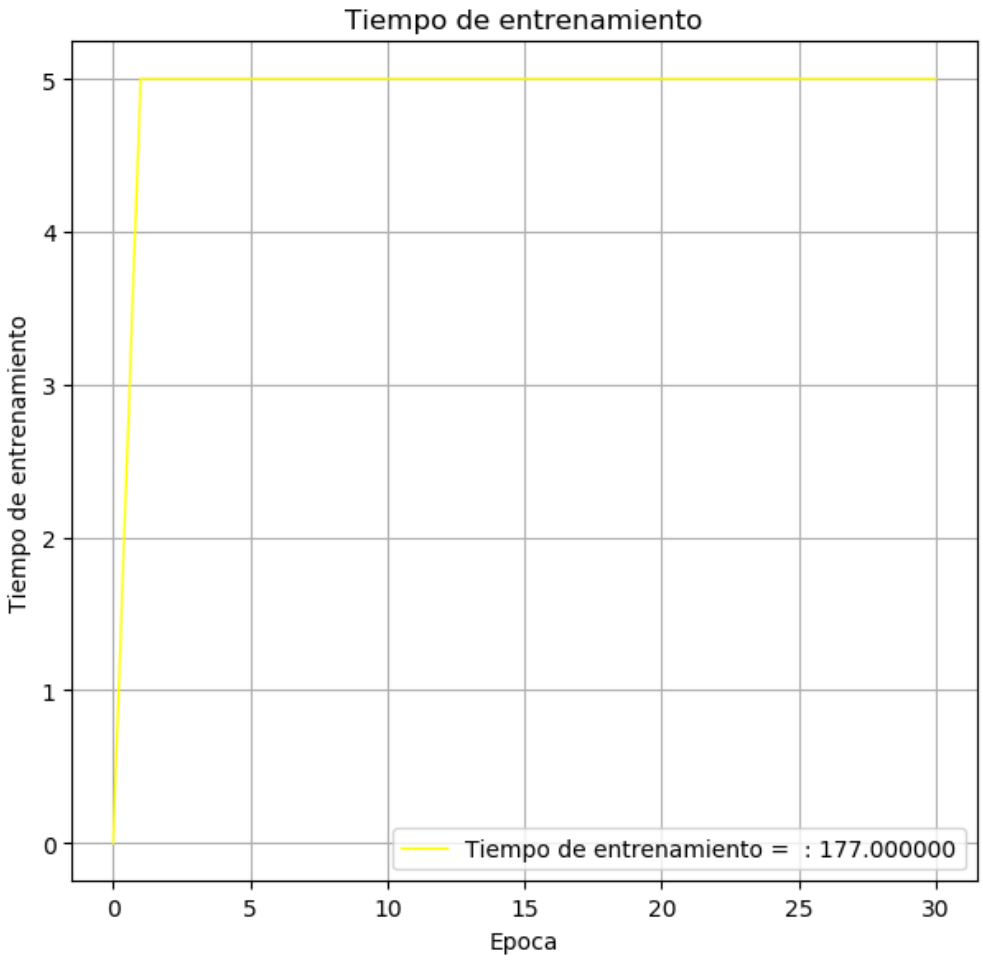

Figura 4.11: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 12.

Fuente: Creación propia

De la Tabla 4.3 que corresponde a entrenamientos utilizando una iteración de 3, del cual observamos que el resultado final para la precisión(accuracy) es 32.77, en cuanto a la pérdida(training loss)es de 2.81 y el tiempo de entrenamiento fue de 2 horas, 57 minutos.

Iteración 3	Tasa de Crecimiento 12
Precisión(Accuracy)	32.77 %
Pérdida(Training Loss)	2.81 %
Tiempo	2 horas, 57 minutos

Tabla 4.3: Resultados para una iteracion de 3.

Fuente: Creación propia

4.3.1.2. Medición de las métricas con iteraciones de 1,2 y 3, con tasas de crecimiento de 32

la medición de las métricas utilizando iteraciones de 1,2 y 3 se detallan en las tablas 4.4, 4.5, 4.6:

Iteracion de 1

De la Figura 4.12 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 70.96 %.

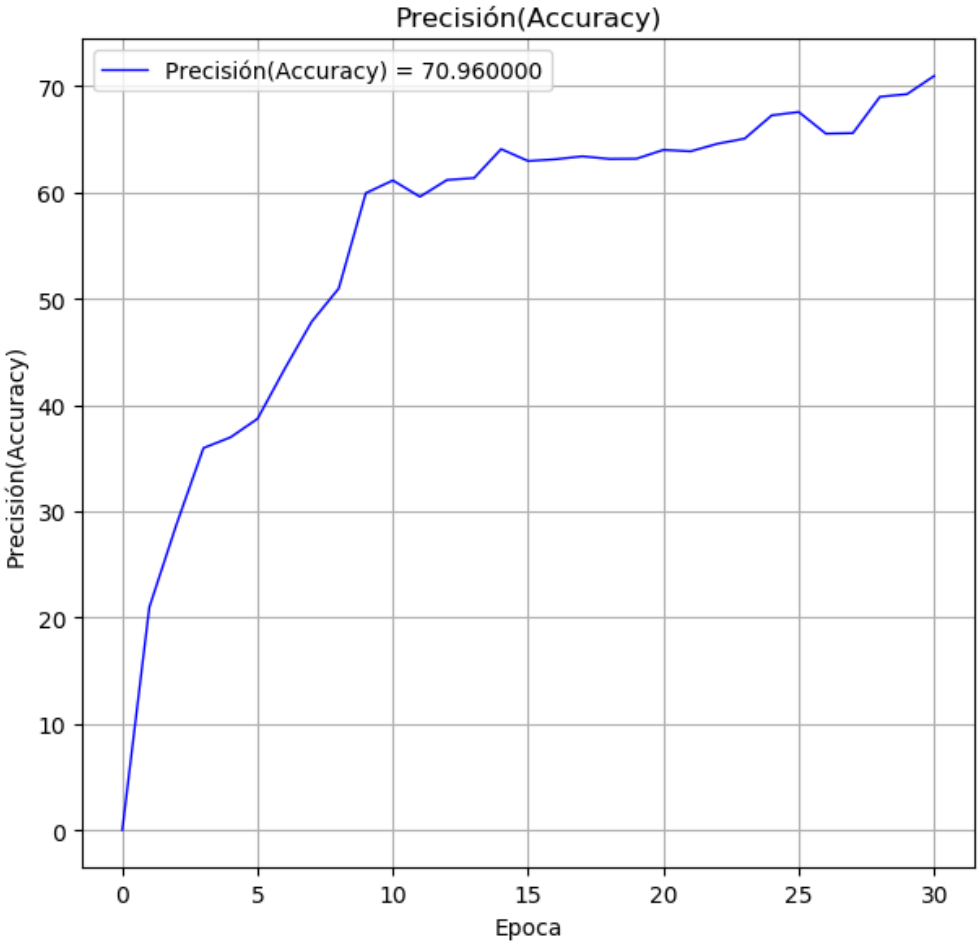

Figura 4.12: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.13 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 2.68 %.

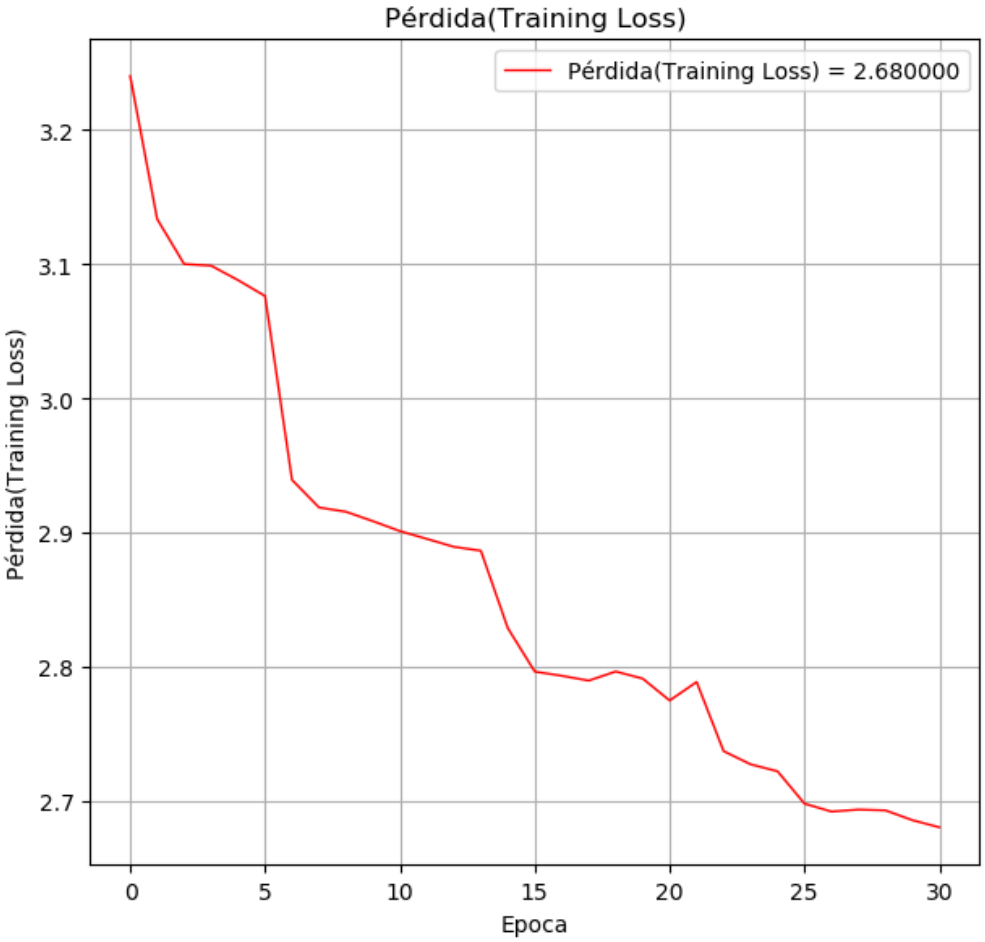

Figura 4.13: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.14 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 3 minutos, realizando un tiempo total de entrenamiento de 1 hora y 47 minutos.

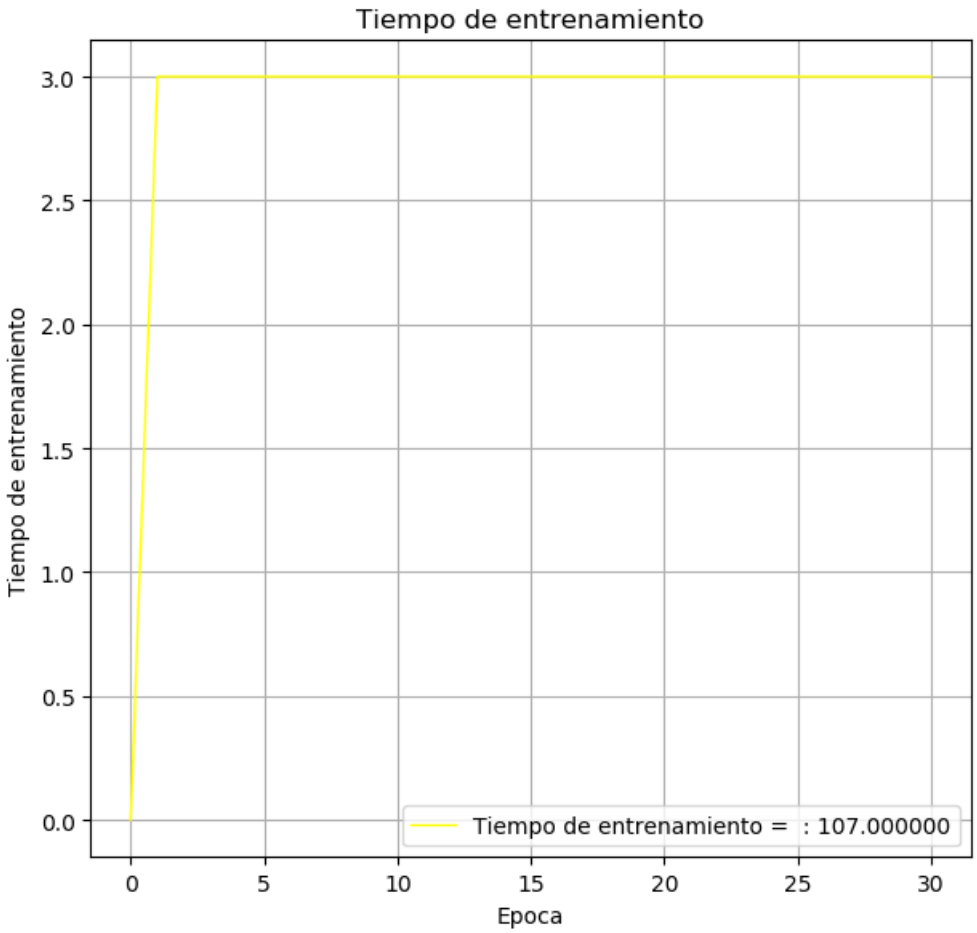

Figura 4.14: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 1 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Tabla 4.4 que corresponde a entrenamientos utilizando una iteración de 1, del cual observamos que el resultado final para la precisión(accuracy) es 70.96, en cuanto a la pérdida(training loss)es de 2.68 y el tiempo de entrenamiento fue de 1 horas, 47 minutos.

Iteración 1	Tasa de Crecimiento 32
Precisión(Accuracy)	70.96 %
Pérdida(Training Loss)	2.68 %
Tiempo	1 hora, 47 minutos

Tabla 4.4: Resultados para una iteracion de 1.

Fuente: Creación propia

Iteracion de 2

De la Figura 4.15 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época. Obteniendo como resultado final una precisión del 94.27 %.

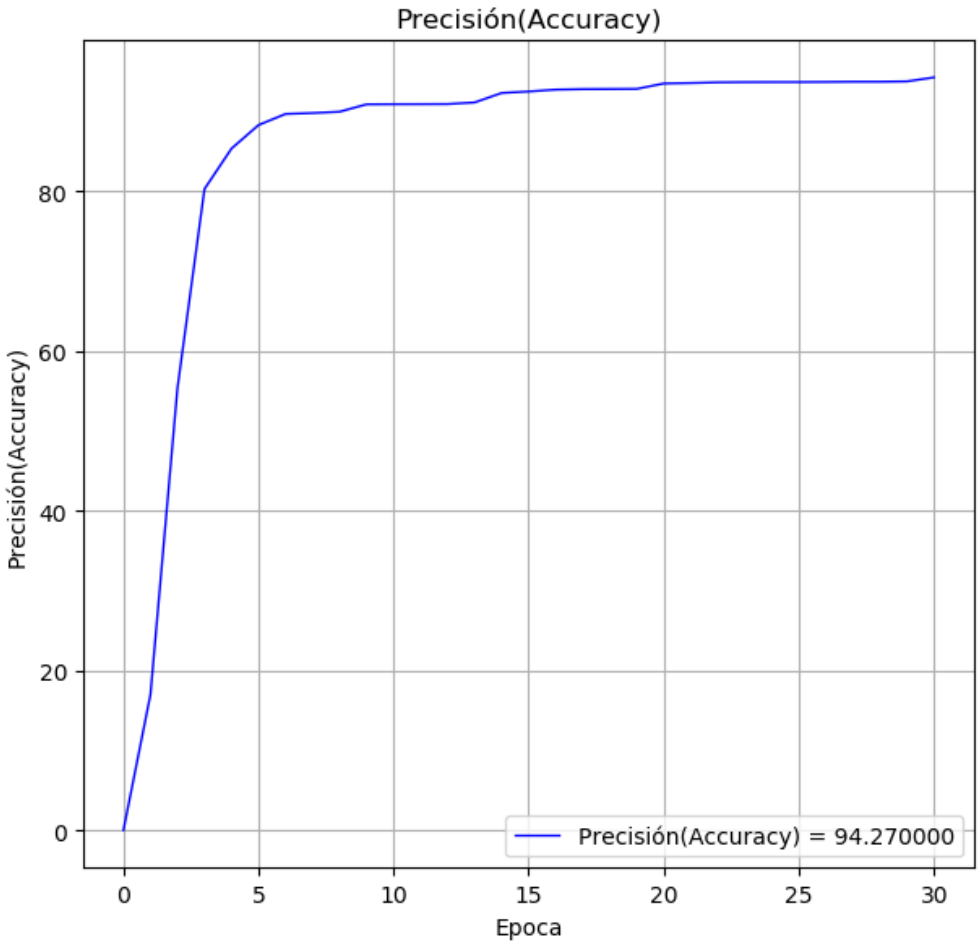

Figura 4.15: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.16 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 0.72 %.

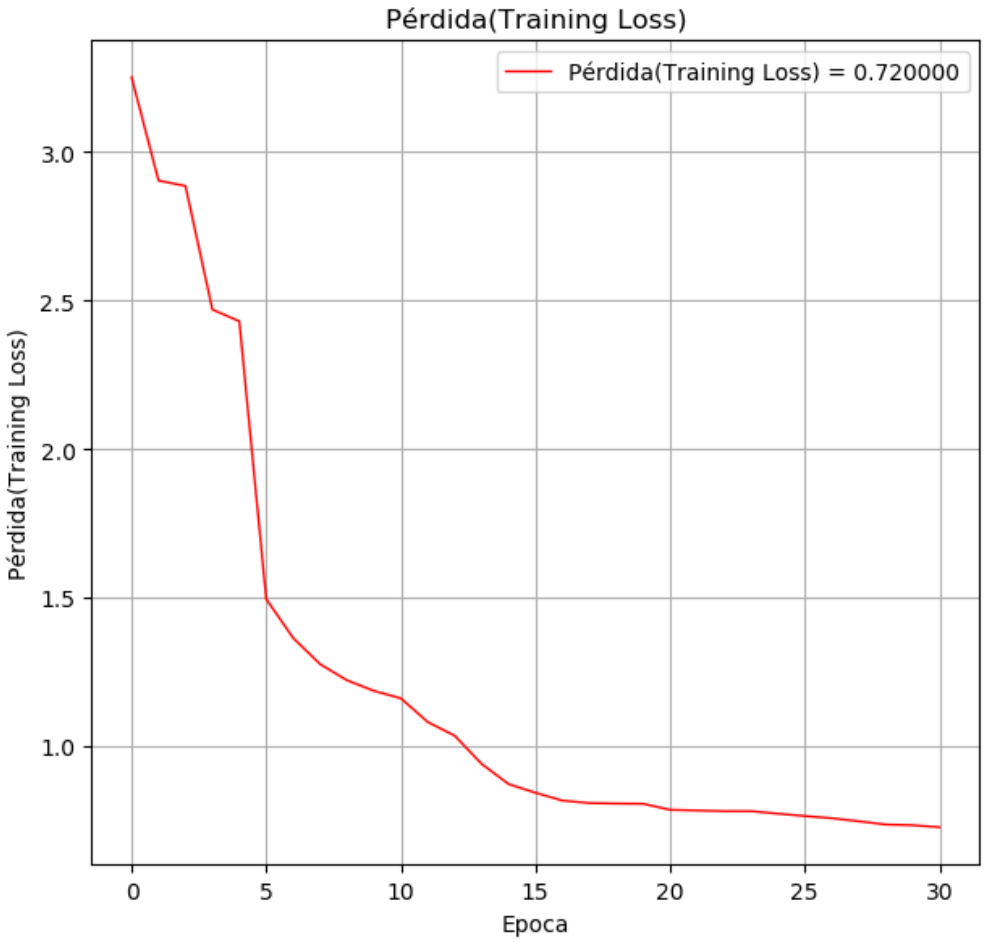

Figura 4.16: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.17 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 4 minutos, realizando un tiempo total de entrenamiento de 2 horas y 8 minutos.

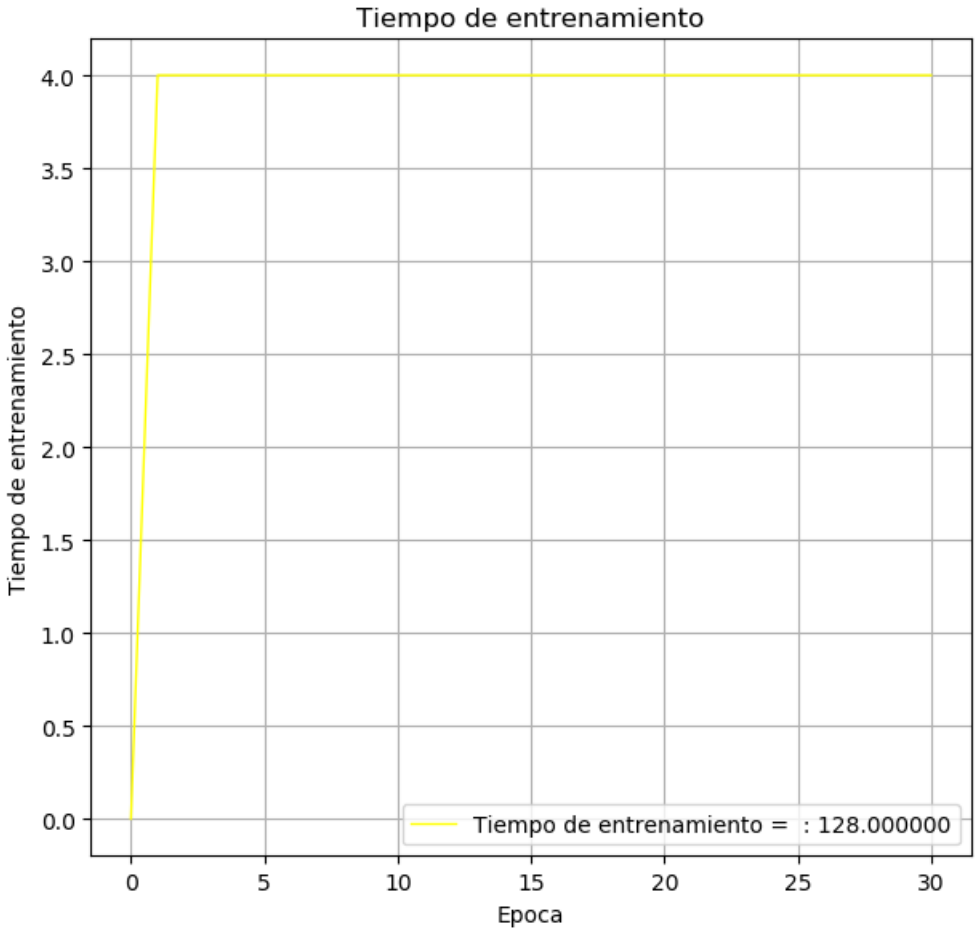

Figura 4.17: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 2 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Tabla 4.5 que corresponde a entrenamientos utilizando una iteración de 2, del cual observamos que el final para la precisión(accuracy) es 94.27, en cuanto a la pérdida(training loss)es de 0.72 y el tiempo de entrenamiento fue de 2 horas, 8 minutos.

Iteración 2	Tasa de Crecimiento 32
Precisión(Accuracy)	94.27 %
Pérdida(Training Loss)	0.72 %
Tiempo	2 horas, 8 minutos

Tabla 4.5: Resultados para una iteracion de 2.

Fuente: Creación propia

Iteracion de 3

De la Figura 4.18 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 52.27 %.

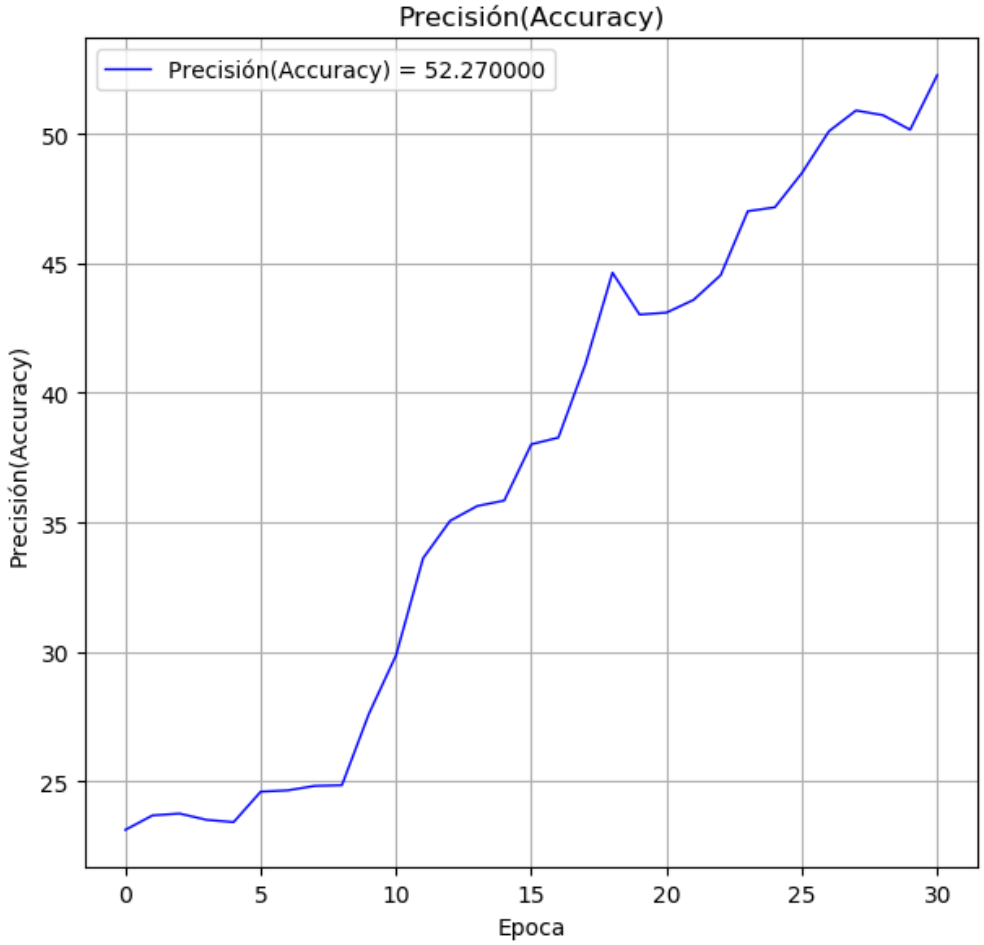

Figura 4.18: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.19 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 2.18 %.

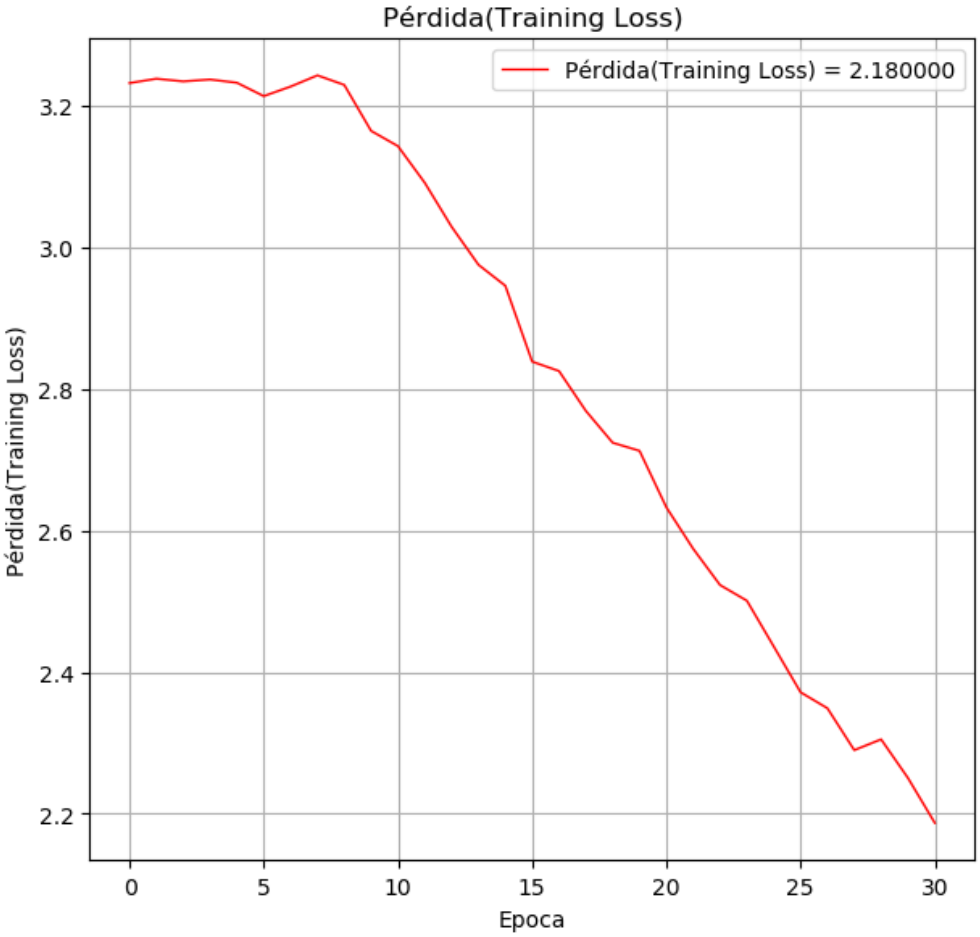

Figura 4.19: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Figura 4.20 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 5 minutos, realizando un tiempo total de entrenamiento de 2 horas y 36 minutos.

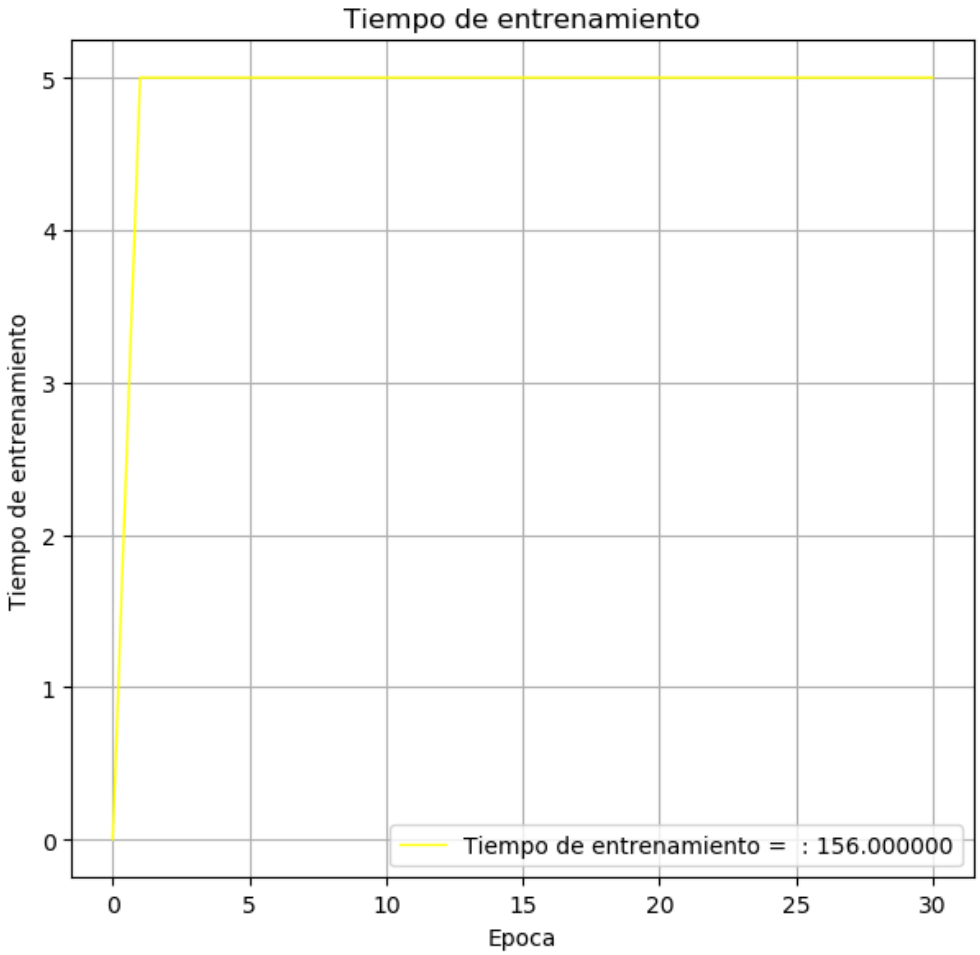

Figura 4.20: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 3 y una tasa de crecimiento de 32.

Fuente: Creación propia

De la Tabla 4.6 que corresponde a entrenamientos utilizando una iteración de 1, del cual observamos que el resultado final para la precisión(accuracy) es 52.27, en cuanto a la pérdida(training loss)es de 2.18 y el tiempo de entrenamiento fue de 2 horas, 36 minutos.

Iteración 3	Tasa de Crecimiento 32
Precisión(Accuracy)	52.27 %
Pérdida(Training Loss)	2.18 %
Tiempo	2 horas, 36 minutos

Tabla 4.6: Resultados para una iteracion de 3.

Fuente: Creación propia

4.3.2. Modelo Base

4.3.2.1. Medición de las métricas con iteraciones de 1,2 y 3

la medición de las métricas utilizando iteraciones de 1,2 y 3 se detallan en las tablas 4.7 ,4.8, 4.9:

Iteracion de 1

De la Figura 4.21 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 75.94 %.

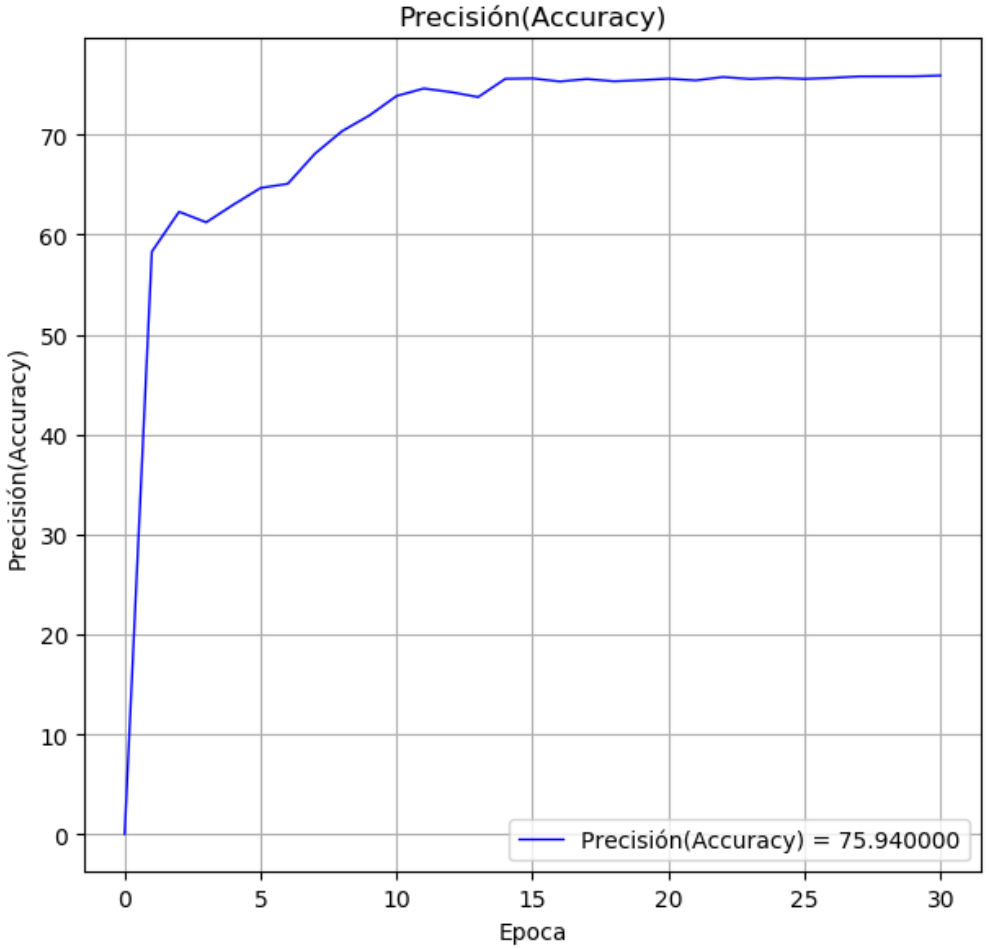

Figura 4.21: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 1.

Fuente: Creación propia

De la Figura 4.22 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 1.97 %.

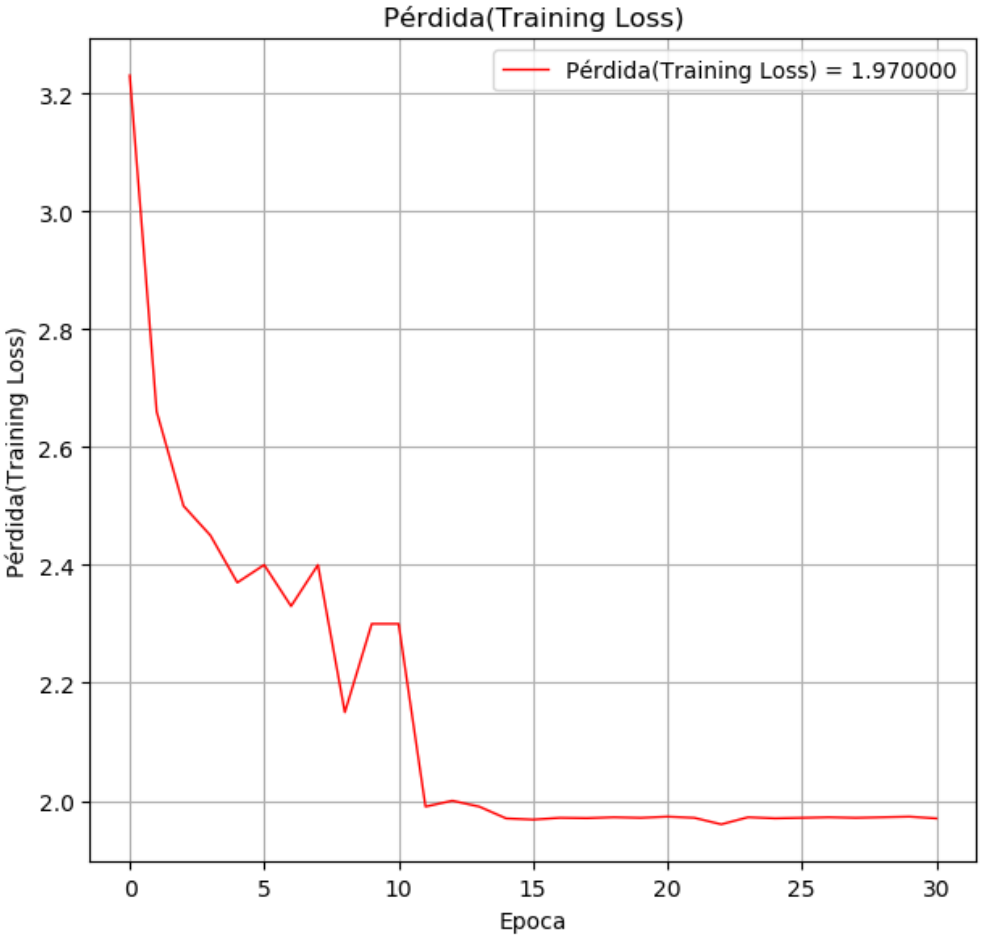

Figura 4.22: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 1.

Fuente: Creación propia

De la Figura 4.23 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 10 minutos, realizando un tiempo total de entrenamiento de 5 horas y 35 minutos.

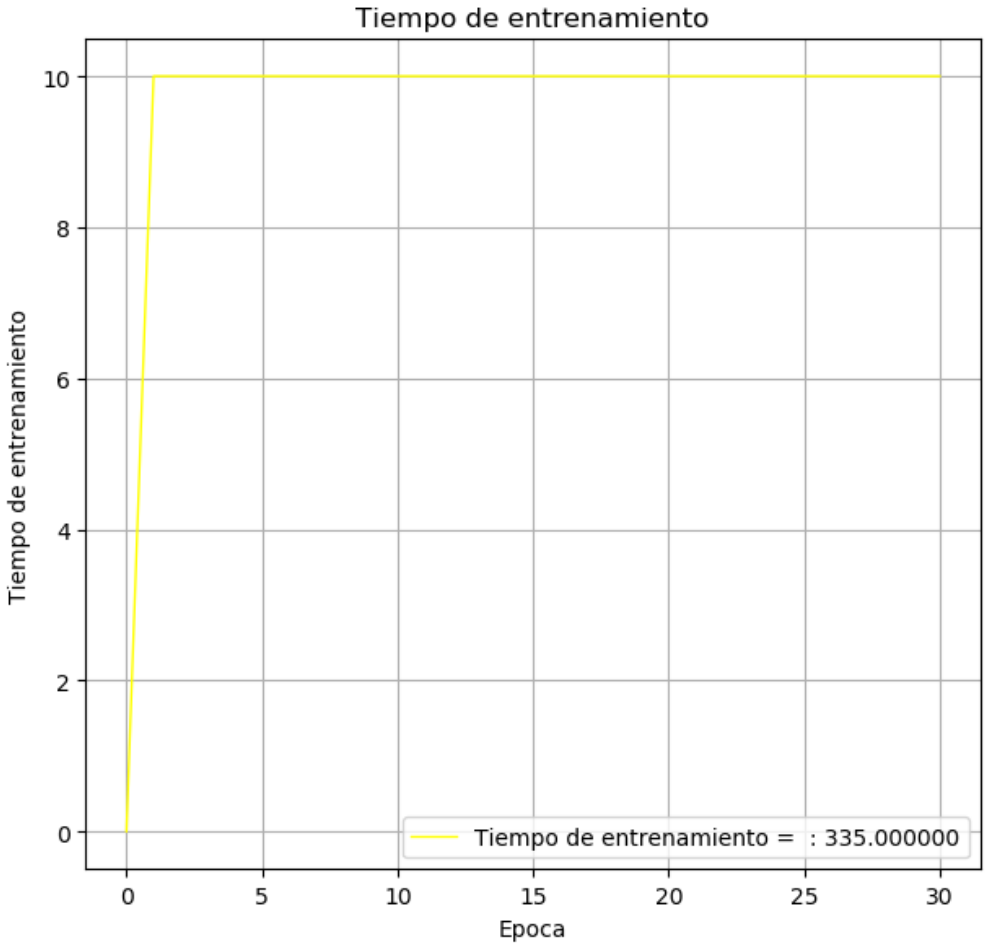

Figura 4.23: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 1.

Fuente: Creación propia

De la Tabla 4.7 que corresponde a entrenamientos utilizando una iteración de 1, del cual observamos que el resultado final para la precisión(accuracy) es 75.94, en cuanto a la pérdida(training loss)es de 1.97 y el tiempo de entrenamiento fue de 5 horas, 35 minutos.

Iteración 1	resultados
Precisión(Accuracy)	75.94 %
Pérdida(Training Loss)	1.97 %
Tiempo	5 horas, 35 minutos

Tabla 4.7: Resultados para una iteracion de 1.

Fuente: Creación propia

Iteracion de 2

De la Figura 4.24 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 89.45 %.

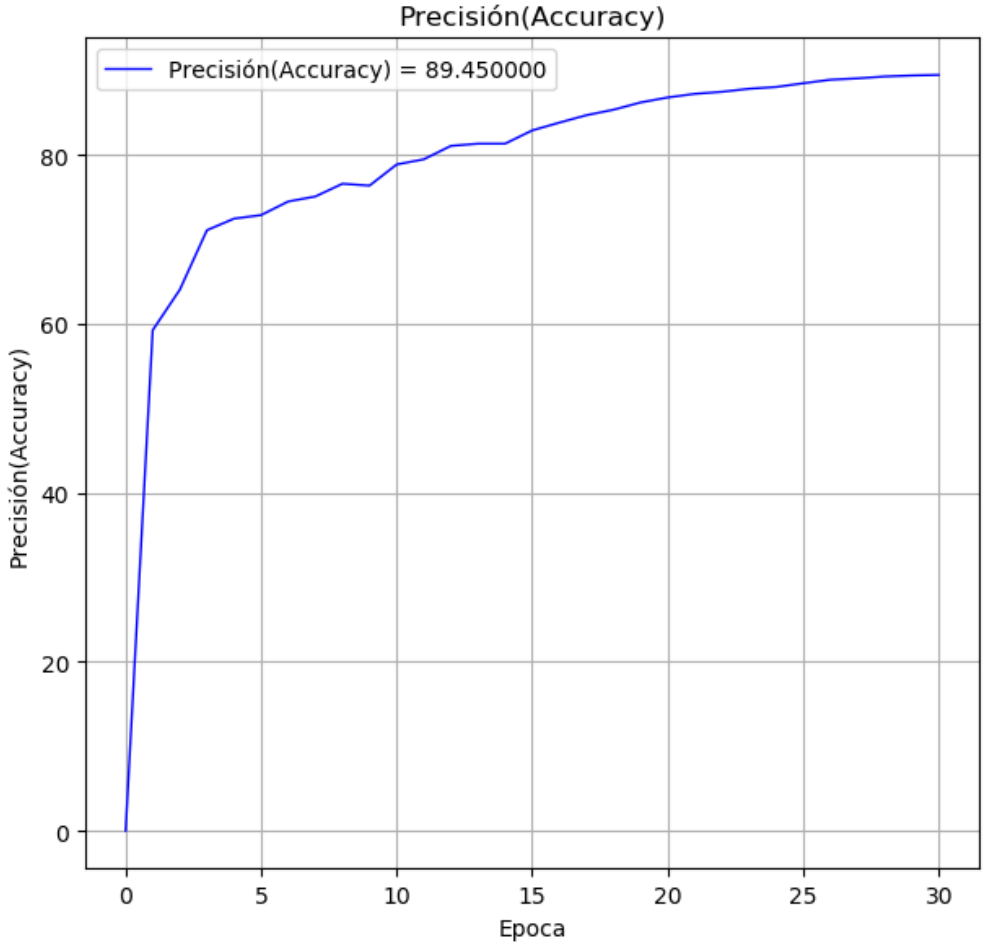

Figura 4.24: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 2.

Fuente: Creación propia

De la Figura 4.25 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 0.91 %.

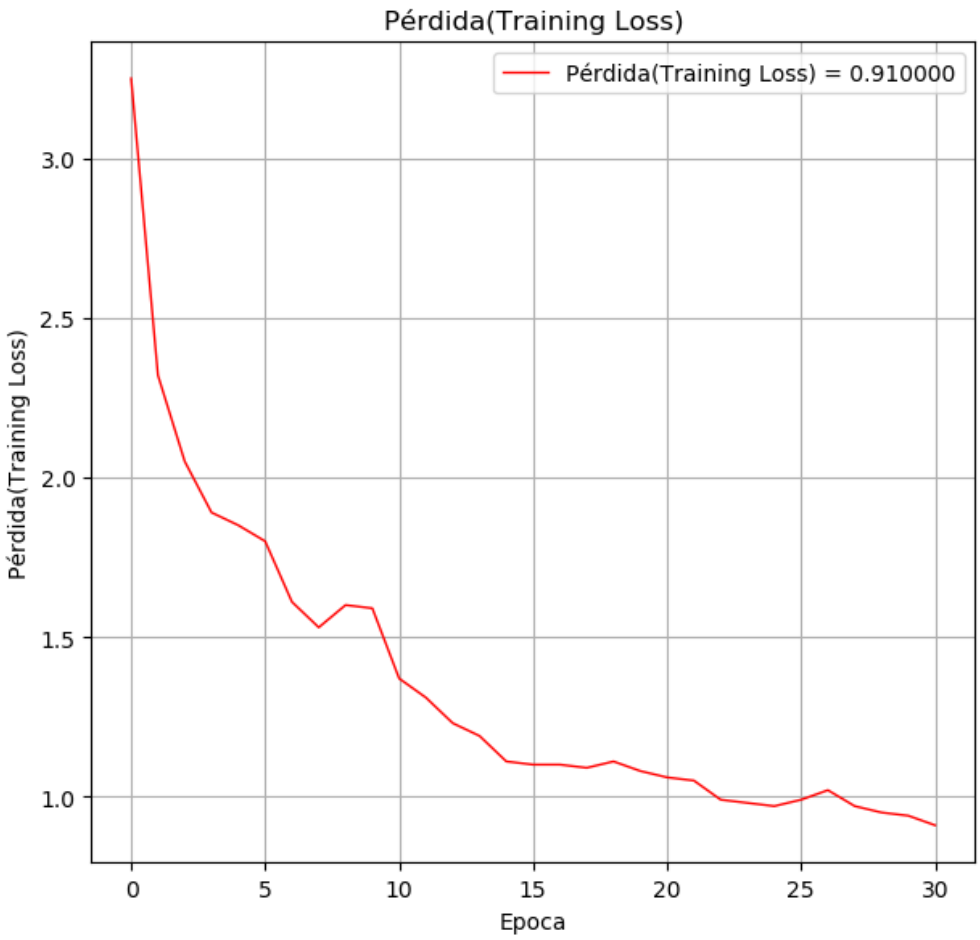

Figura 4.25: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 2.

Fuente: Creación propia

De la Figura 4.26 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 15 minutos, realizando un tiempo total de entrenamiento de 7 horas y 43 minutos.

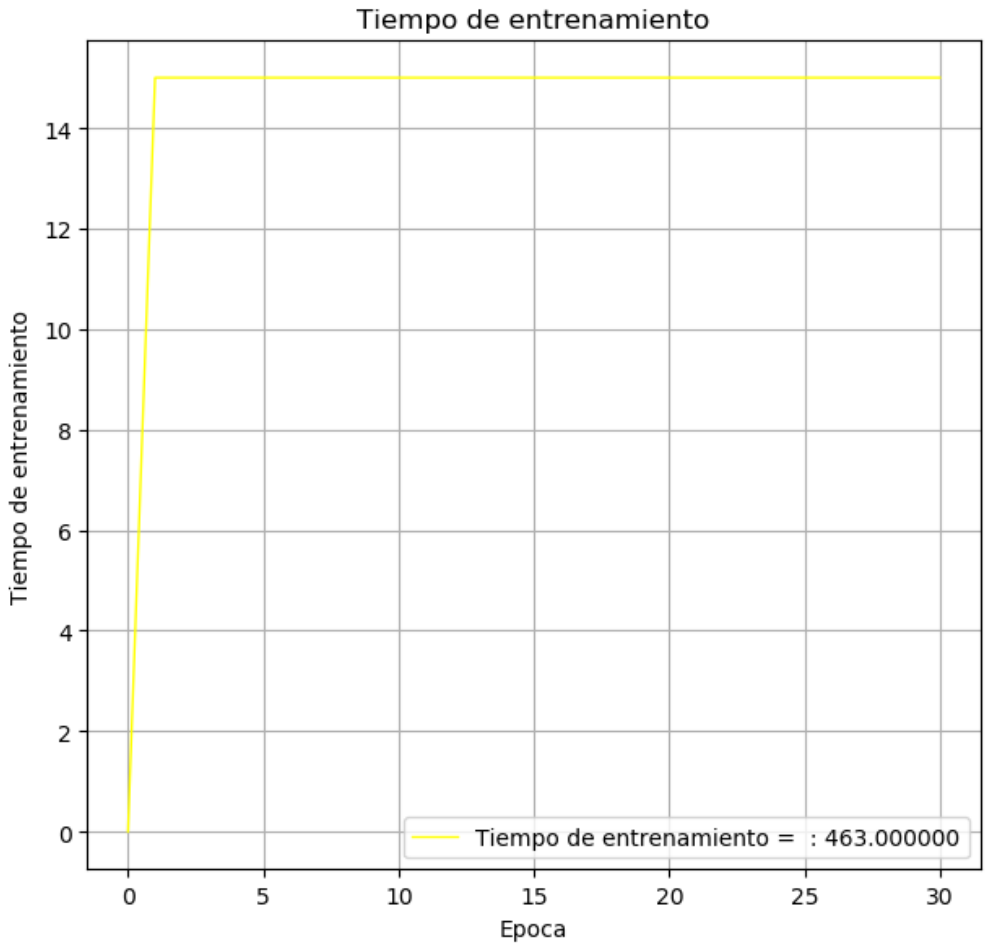

Figura 4.26: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 2.

Fuente: Creación propia

De la Tabla 4.8 que corresponde a entrenamientos utilizando una iteración de 2, del cual observamos que el resultado final para el accuracy es 89.45, en cuanto al Average loss es de 0.91 y el tiempode entrenamiento fue de 7 horas, 43 minutos.

Iteración 2	Resultados
Precisión(Accuracy)	89.45 %
Pérdida(Training Loss)	0.91 %
Tiempo	7 horas, 43 minutos

Tabla 4.8: Resultados para una iteracion de 2.

Fuente: Creación propia

Iteracion de 3

De la Figura 4.27 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar un crecimiento ascendente de los resultados obtenidos en cada época salvo en algunas épocas donde hubo un pequeño decremento. Obteniendo como resultado final una precisión del 66.25 %.

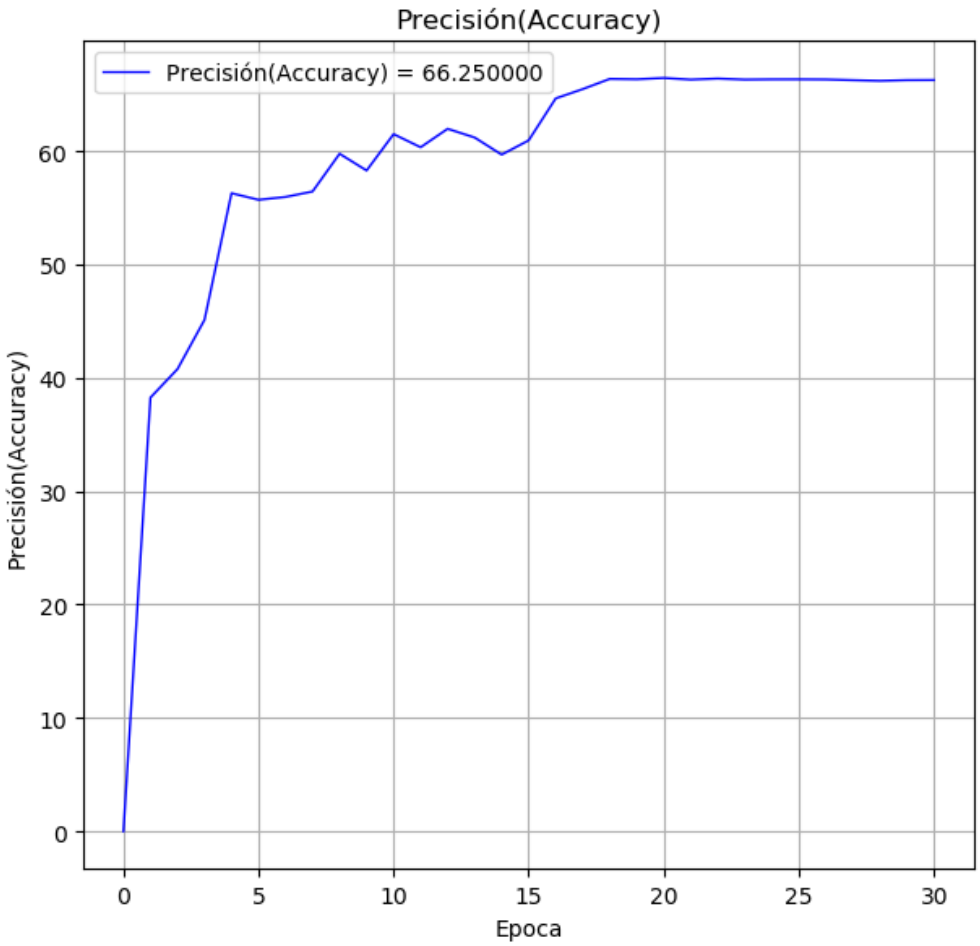

Figura 4.27: Medición de la precisión en un entrenamiento de 30 epocas, para una iteracion de 3.

Fuente: Creación propia

De la Figura 4.28 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un crecimiento descendente de los resultados obtenidos en cada época, obteniendo un resultado final de pérdida de entrenamiento del 1.61 %.

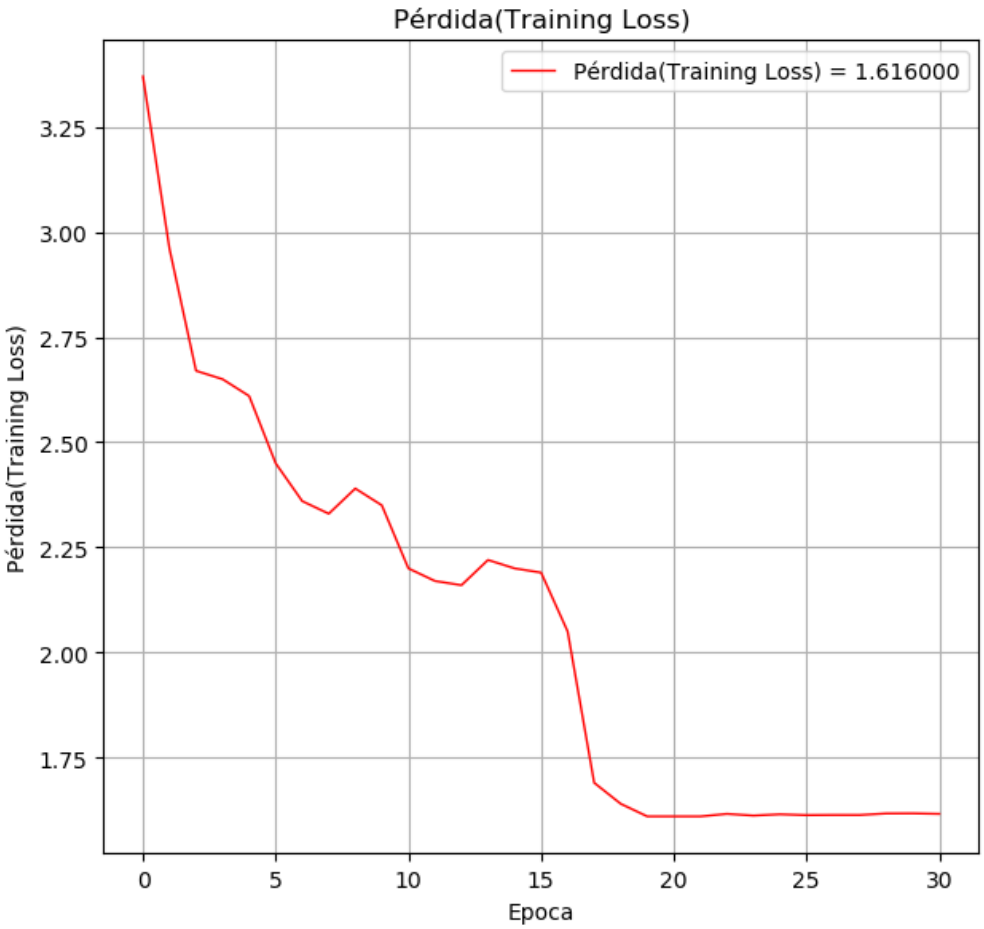

Figura 4.28: Medición de la pérdida en un entrenamiento de 30 epocas, para una iteracion de 3.

Fuente: Creación propia

De la Figura 4.29 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar un tiempo entramiento constante, es decir, el tiempo que se emplea en entrenar una época es de 20 minutos, realizando un tiempo total de entrenamiento de 10 horas y 8 minutos.

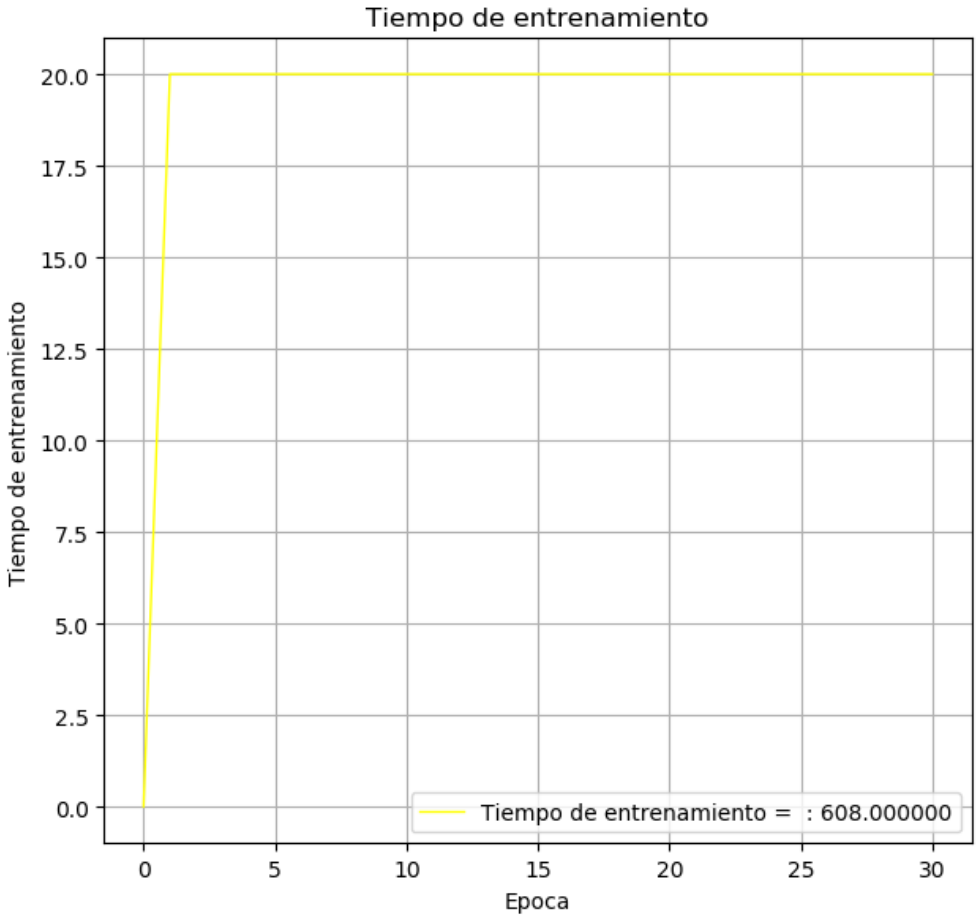

Figura 4.29: Medición de la tiempo en un entrenamiento de 30 epocas, para una iteracion de 3.

Fuente: Creación propia

De la Tabla 4.9 que corresponde a entrenamientos utilizando una iteración de 3, del cual observamos que el resultado final para la precisión(accuracy) es 66.25, en cuanto a la pérdida(training loss)es de 1.61 y el tiempo de entrenamiento fue de 10 horas, 8 minutos.

Iteración 3	Resultados
Precisión(Accuracy)	66.25 %
Pérdida(Training Loss)	1.61 %
Tiempo	10 horas, 8 minutos

Tabla 4.9: Resultados para una iteracion de 3.

Fuente: Creación propia

Parte V

Análisis de Resultados

Capítulo 5

Resultados

De la Figura 5.1 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar que el modelo base es superior al modelo propuesto con tasas de crecimiento de 12 y 32, obteniendo como resultado una precisión de 75.94 % contra un 67.72 % y 70.96 %respectivamente. Por lo tanto, se deduce que si se utiliza una iteración de 1 el modelo base es más eficaz que el modelo propuesto.

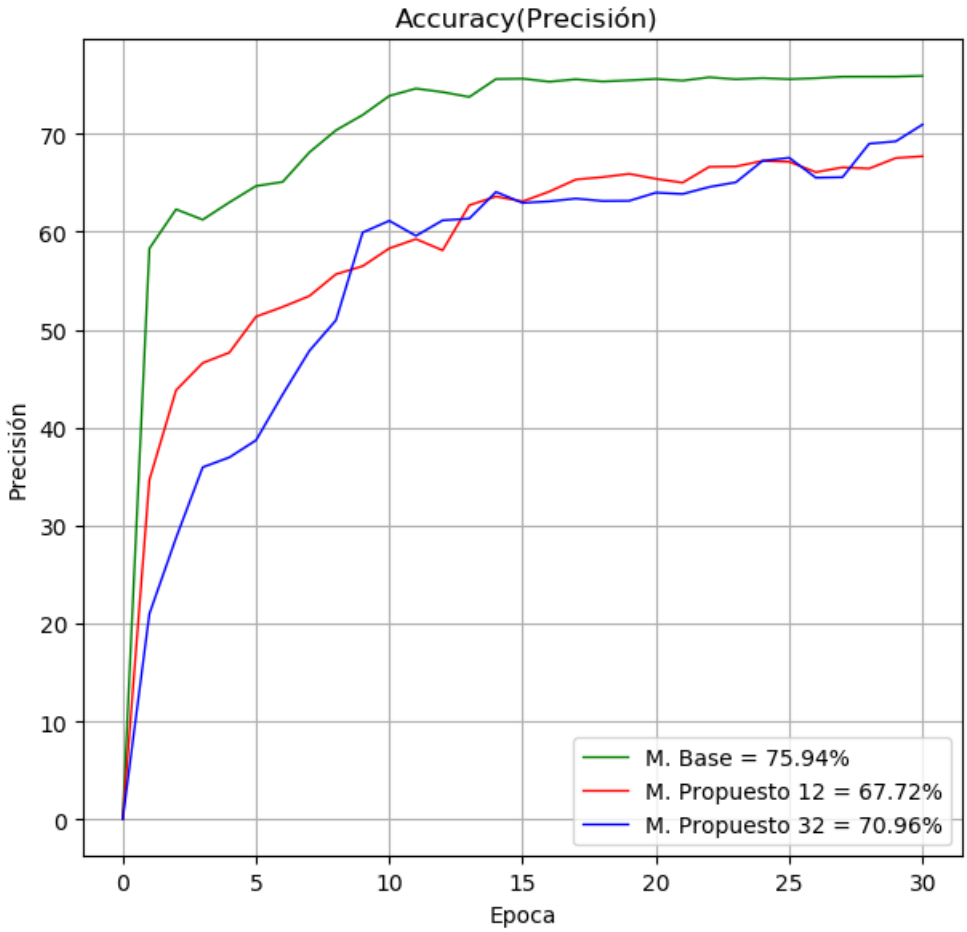

Figura 5.1: Resultados para la precisión (Accuracy) con una iteración de 1,
Fuente: Creación propia

De la Figura 5.2 se observa los resultados con respecto a la precisión (Accu-

racy), donde se puede apreciar que el modelo propuesto es superior al modelo base, ya sea utilizando tasas de crecimiento de 12 y 32, obteniendo como mejor resultado una precisión de 94.27 % si se utiliza una tasa de crecimiento de 32 frente al 89.45 % del modelo base. Por lo tanto, se deduce que si se utiliza una iteración de 2 el modelo propuesto es más eficaz que el modelo base.

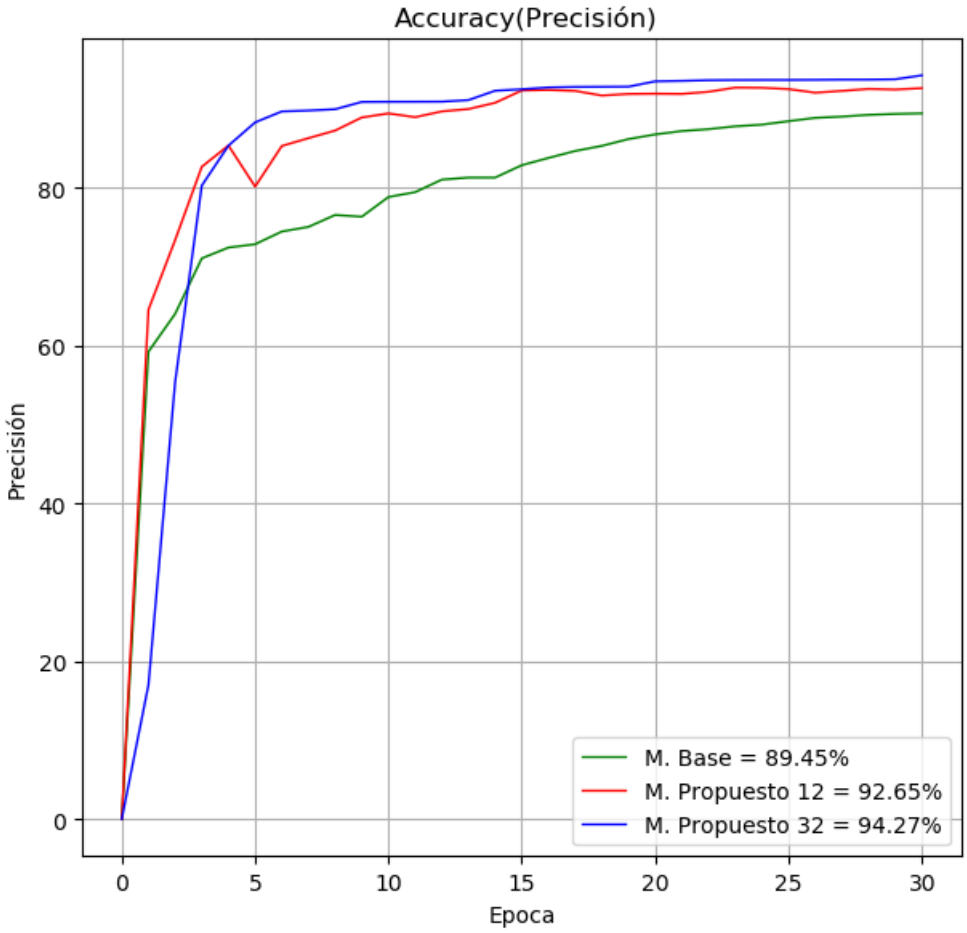

Figura 5.2: Resultados para la precisión (Accuracy) con una iteración de 2,
Fuente: Creación propia

De la Figura 5.3 se observa los resultados con respecto a la precisión (Accuracy), donde se puede apreciar que el modelo base es superior al modelo propuesto con tasas de crecimiento de 12 y 32, obteniendo como resultado una precisión de 66.25 % contra un 32.77 % y 52.27 % respectivamente. Por lo tanto, se deduce que si se utiliza una iteración de 3 el modelo base es más eficaz que el modelo propuesto.

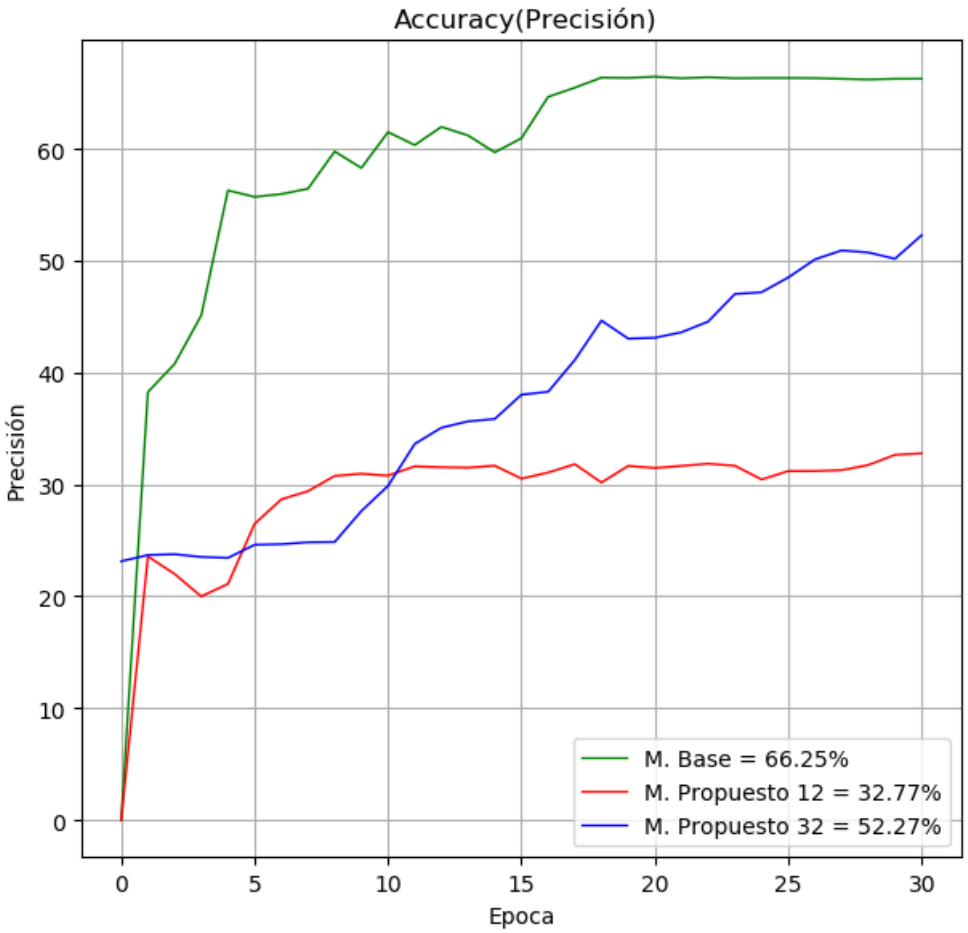

Figura 5.3: Resultados para la precisión (Accuracy) con una iteración de 3,
Fuente: Creación propia

En la tabla 5.1 se muestra la comparacion de los resultados obtenidos con respecto al Accuracy del modelo propuesto y del modelobase.

	Precisión(Accuracy)		
	Iteración 1	Iteración 2	Iteración 3
Modelo Propuesto - tasa de crecimiento 12	67.72 %	92.65 %	32.77 %
Modelo Propuesto - tasa de crecimiento 32	70.96 %	94.27 %	52.27 %
Modelo Base	75.94 %	89.45 %	66.25 %

Tabla 5.1: Resultados para el Accuracy.
Fuente: Creación propia

De la Figura 5.4 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar que el modelo base obtiene un menor porcentaje de pérdida con respecto al modelo propuesto con tasas de crecimiento de 12 y 32, obteniendo como resultado una pérdida de 1.97 % contra un 2.75 % y 2.68 % respectivamente. Por lo tanto, se deduce que si se utiliza una iteración de 1 el modelo base presenta un menor porcentaje de pérdida que el modelo propuesto.

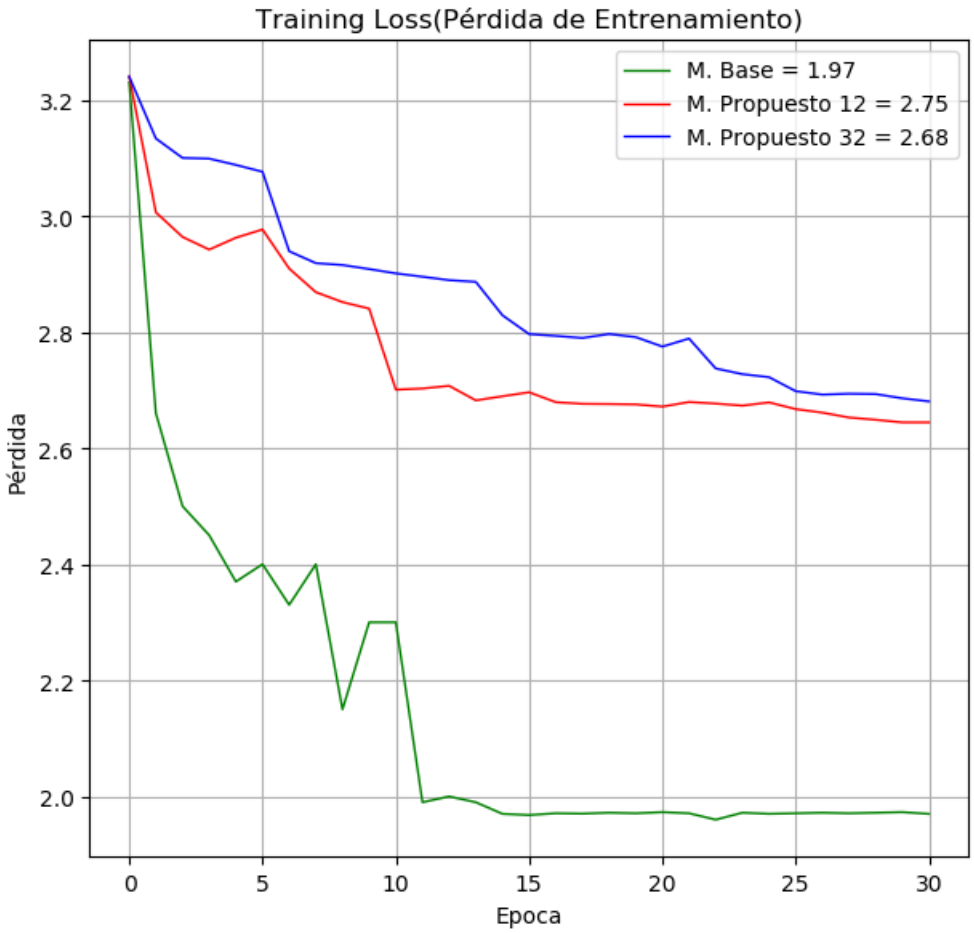

Figura 5.4: Resultados para la pérdida de entrenamiento (Training Loss) con una iteración de 1,

Fuente: Creación propia

De la Figura 5.5 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar que el modelo propuesto obtiene un menor porcentaje de pérdida con respecto al modelo base, ya sea utilizando tasas de crecimiento de 12 y 32, obteniendo como mejor resultado una pérdida de 0.72 % si se utiliza una tasa de crecimiento de 32 frente al 0.91 % del modelo base. Por lo tanto, se deduce que si se utiliza una iteración de 2 el modelo propuesto presenta un menor porcentaje de pérdida que el modelo base.

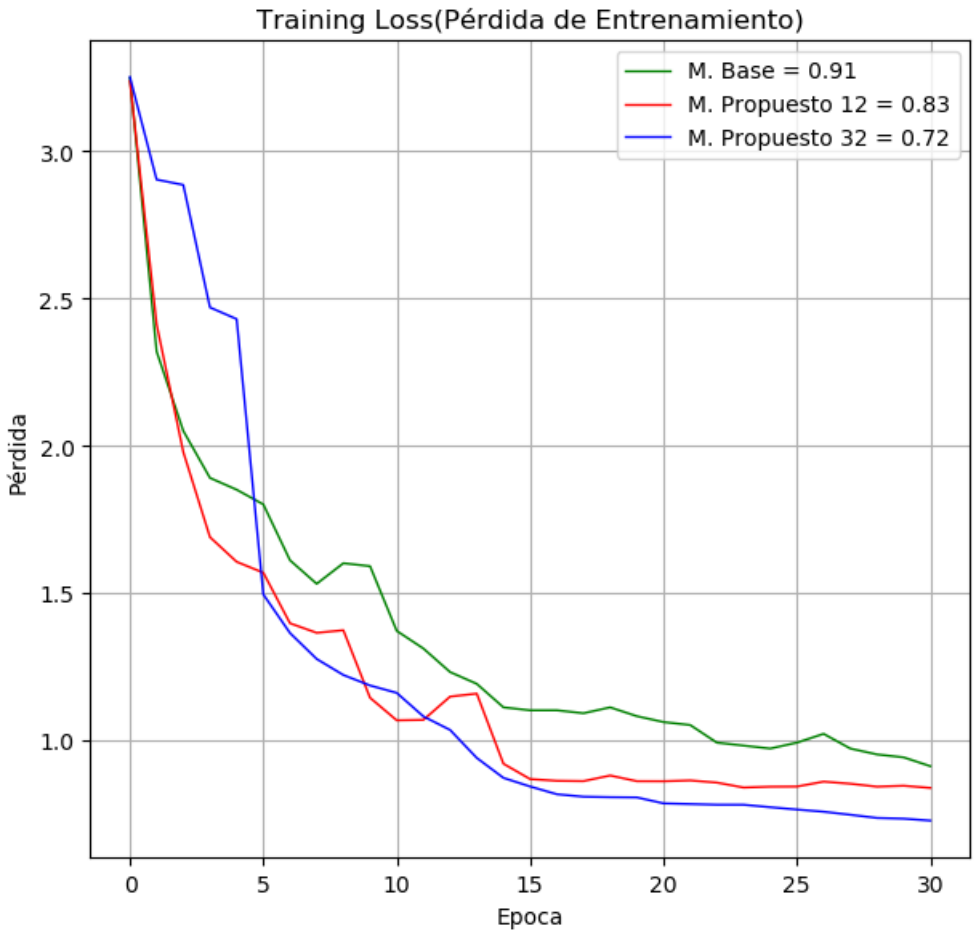

Figura 5.5: Resultados para la pérdida de entrenamiento (Training Loss) con una iteración de 2,

Fuente: Creación propia

De la Figura 5.6 se observa los resultados con respecto a la pérdida de entrenamiento (Training Loss), donde se puede apreciar que el modelo base obtiene un menor porcentaje de pérdida con respecto al modelo propuesto con tasas de crecimiento de 12 y 32, obteniendo como resultado una pérdida de 1.61 % contra un 2.81 % y 2.18 % respectivamente. Por lo tanto, se deduce que si se utiliza una iteración de 1 el modelo base presenta un menor porcentaje de pérdida que el modelo propuesto.

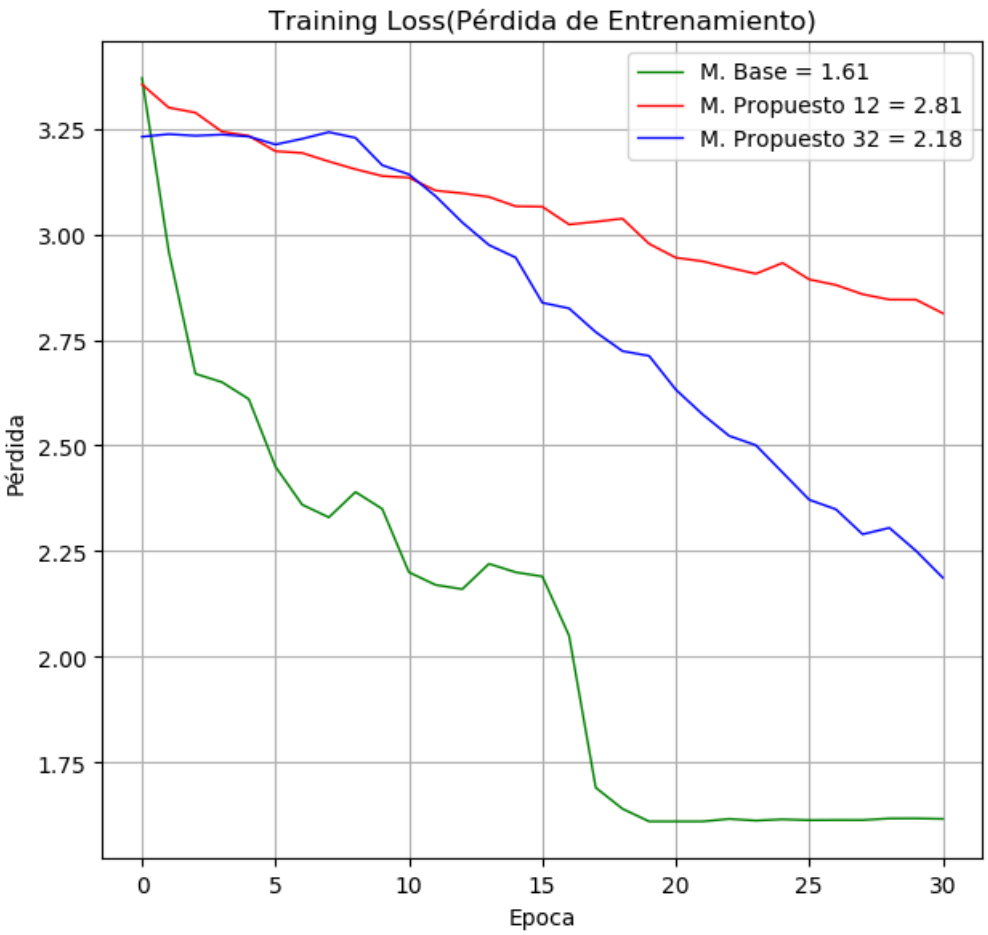

Figura 5.6: Resultados para la pérdida de entrenamiento (Training Loss) con una iteración de 3,

Fuente: Creación propia

En la tabla 5.2 se muestra la comparacion de los resultados obtenidos con respecto al Accuracy del modelo propuesto y del modelobase.

	Training Loss		
	Iteración 1	Iteración 2	Iteración 3
Modelo Propuesto - tasa de crecimiento 12	2.75 %	0.83 %	2.81 %
Modelo Propuesto - tasa de crecimiento 32	2.68 %	0.72 %	2.18 %
Modelo Base	1.97 %	0.91 %	1.61 %

Tabla 5.2: Resultados para la Pérdida de Entrenamiento(Training Loss).

Fuente: Creación propia

De la Figura 5.7 se observa los resultados con respecto al tiempo de entrenamiento, donde se observa que el modelo propuesto con una tasa de crecimiento de 32 presenta un menor tiempo de entrenamiento frente al modelo base y al mismo modelo propuesto pero con una tasa de crecimiento de 12, obteniendo como resultado un tiempo de entrenamiento de 1 hora con 47 minutos frente a las 5 horas con 35 minutos del modelo propuesto y las 2 horas con 13 minutos del modelo propuesto con una tasa de crecimiento de 12. Por lo tanto, se deduce que si se

utiliza una iteración de 1 el modelo propuesto presentara un menor tiempo de entrenamiento.

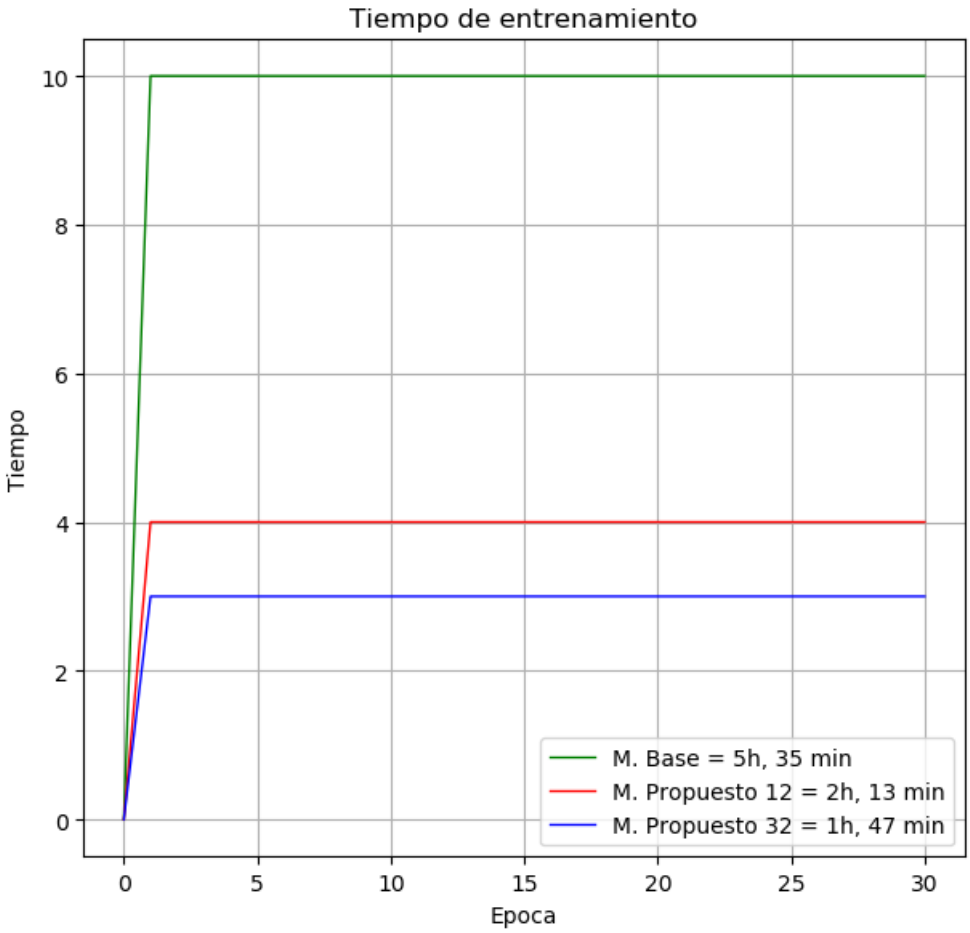

Figura 5.7: Resultados para el tiempo de entrenamiento con una iteración de 1,
Fuente: Creación propia

De la Figura 5.8 se observa los resultados con respecto al tiempo de entrenamiento, donde se observa que el modelo propuesto con una tasa de crecimiento de 32 presenta un menor tiempo de entrenamiento frente al modelo base y al mismo modelo propuesto pero con una tasa de crecimiento de 12, obteniendo como resultado un tiempo de entrenamiento de 2 hora con 8 minutos frente a las 7 horas con 43 minutos del modelo propuesto y las 2 horas con 32 minutos del modelo propuesto con una tasa de crecimiento de 12. Por lo tanto, se deduce que si se utiliza una iteración de 1 el modelo propuesto presentara un menor tiempo de entrenamiento.

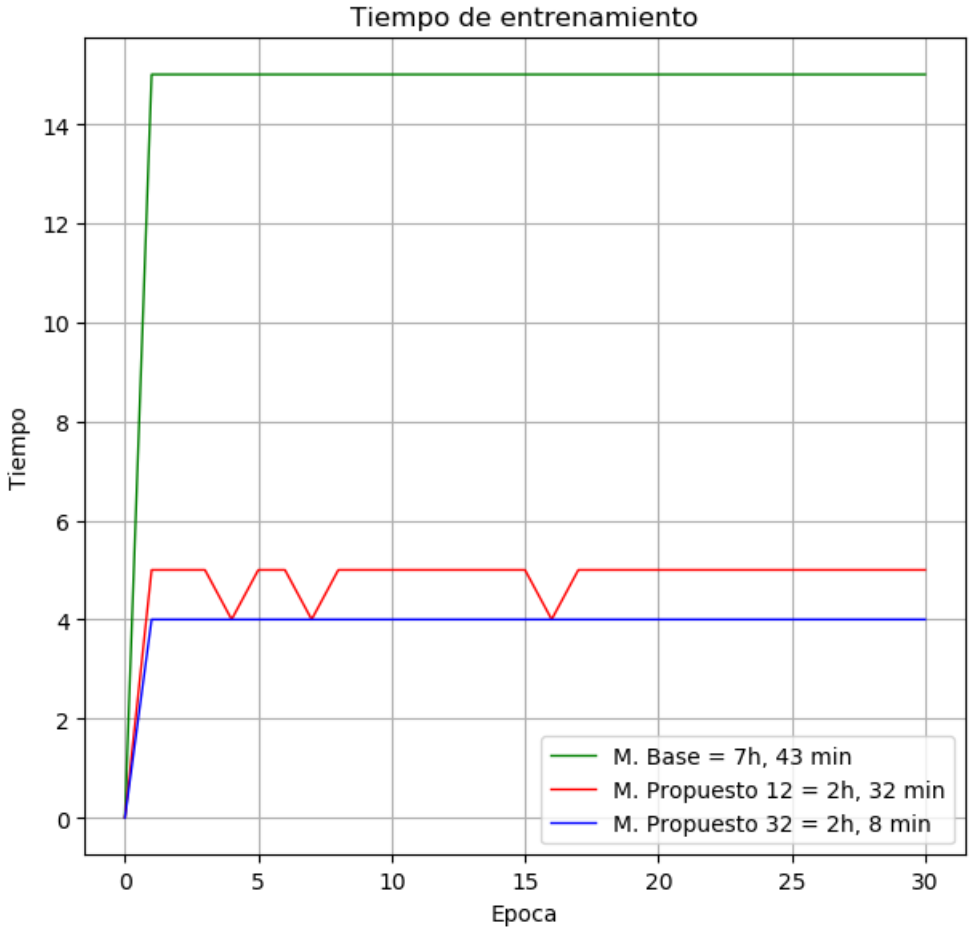

Figura 5.8: Resultados para el tiempo de entrenamiento con una iteración de 2,
Fuente: Creación propia

De la Figura 5.9 se observa los resultados con respecto al tiempo de entrenamiento, donde se observa que el modelo propuesto con una tasa de crecimiento de 32 presenta un menor tiempo de entrenamiento frente al modelo base y al mismo modelo propuesto pero con una tasa de crecimiento de 12, obteniendo como resultado un tiempo de entrenamiento de 2 horas con 36 minutos frente a las 10 horas con 8 minutos del modelo propuesto y las 2 horas con 57 minutos del modelo propuesto con una tasa de crecimiento de 12. Por lo tanto, se deduce que si se utiliza una iteración de 1 el modelo propuesto presentara un menor tiempo de entrenamiento.

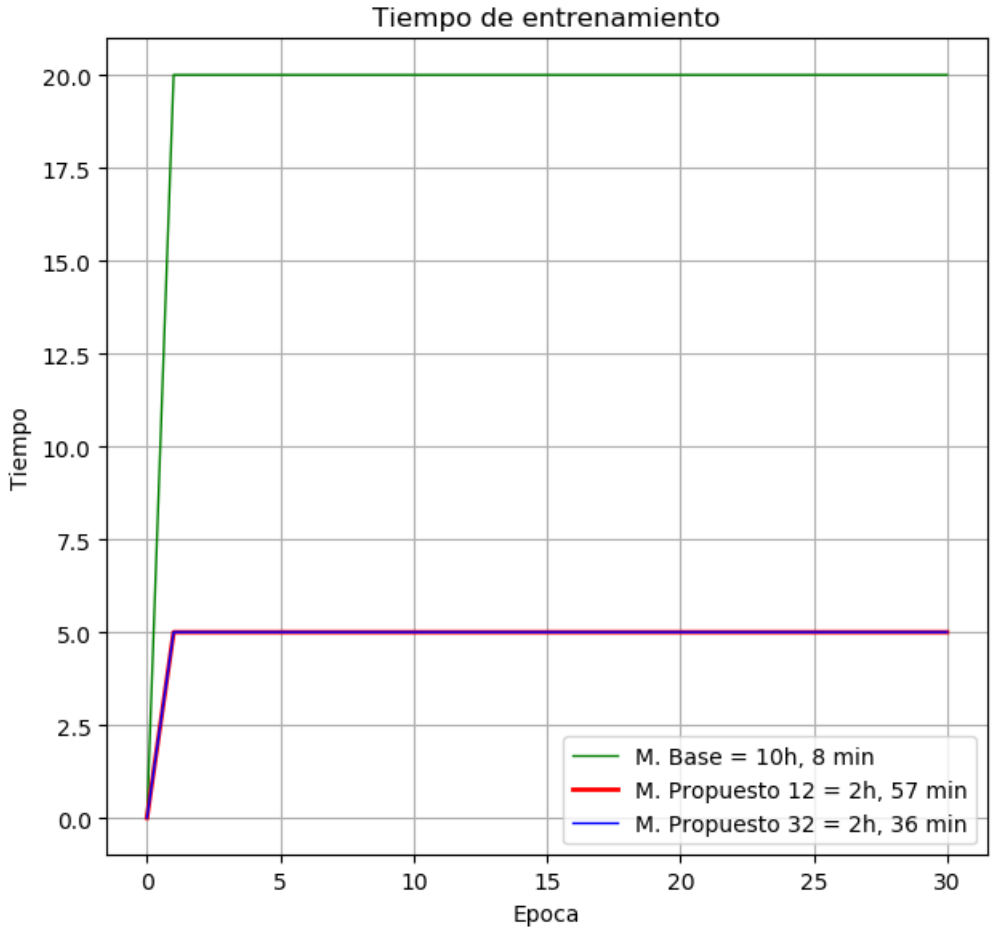

Figura 5.9: Resultados para el tiempo de entrenamiento con una iteración de 3,
Fuente: Creación propia

En la tabla 5.3 se muestra la comparacion de los resultados obtenidos con respecto al tiempo de entrenamiento del modelo propuesto y del modelobase.

	Tiempo de Entrenamiento		
	Iteración 1	Iteración 2	Iteración 3
Modelo Propuesto - tasa de crecimiento 12	2 h, 13 min	2 h, 32 min	2 h, 57 min
Modelo Propuesto - tasa de crecimiento 32	1 h, 47 min	2 h, 8 min	2 h, 36 min
Modelo Base	5 h, 35 min	7 h, 43 min	10 h, 8 min

Tabla 5.3: Resultados para el tiempo.
Fuente: Creación propia

De las tablas 5.1, 5.2 y 5.3 se deduce que el mejor resultado obtenido tomando como prioridad la precisión (Accuracy) es el modelo propuesto con una tasa de crecimiento de 32 y con una iteración de 2, obteniendo los siguientes resultados: Precisión (Accuracy) de 94.27 %, Pérdida de entrenamiento (Training loss) de 0.72 % y tiempo de entrenamiento de 2 horas y 8 minutos.

De las figuras 5.10 y 5.11 se observa que con el modelo propuesto hay una menor exigencia en el GPU con respecto al modelo base.

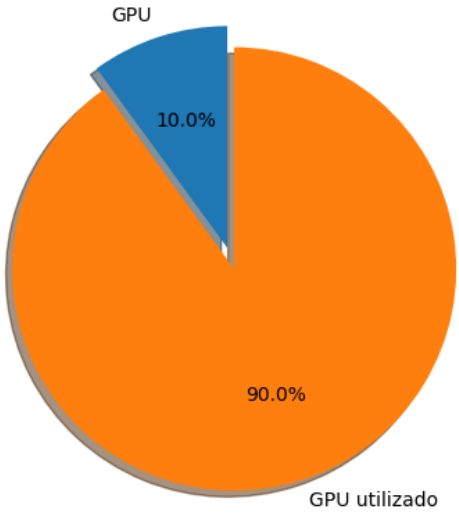

Figura 5.10: Rendimiento del GPU utilizando el Modelo Base.
Fuente: Creación propia

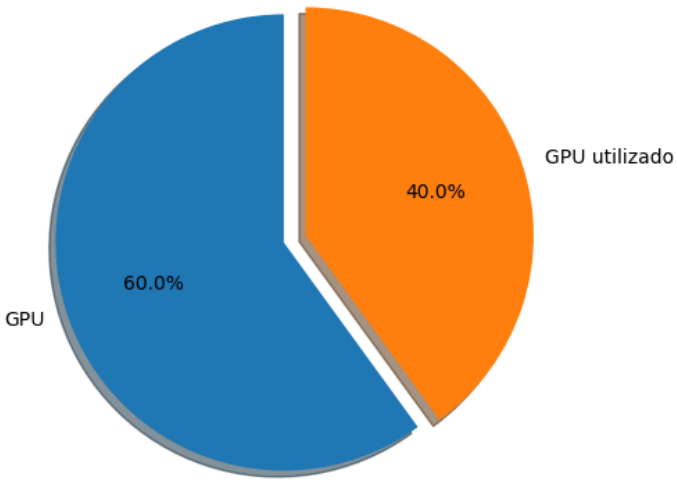

Figura 5.11: Rendimiento del GPU utilizando el Modelo Propuesto.
Fuente: Creación propia

Parte VI

Conclusiones, Recomendaciones y Trabajos Futuros

Capítulo 6

Conclusiones

1. Conclusión en base al objetivo general:

   a) Se diseño un modelo en el cual se reemplaza la capa RELU Conv1 de una Cápsula de matriz con Enrutamiento EM con redes convolucionales densas (Densenet), este proceso ayuda a aprender mejores mapas de características y a su vez se forma capsulas primarias de mejor calidad. La eficacia de este modelo se demuestra por su rendimiento. Tomando como principal métrica de evaluación la precisión(Accuracy), se obtiene como mejor resultado en un entrenamiento de 30 epocas y utilizando el dataset SmallNORB, un 94.27 % de precisión(accuracy), una pérdida de entrenamiento(Training Loss) de 0.72 % en un tiempo de 2 horas y 8 minutos en comparación con el modelo base que obtuvo un 89.45 % de precisión(Acuraccy),una pérdida de entrenamiento(Training Loss) de 0.91 % en un tiempo de 7 horas con 43 minutos.

2. Conclusión en base al primer objetivo especifico:

   a) Se realizo un análisis de la Red Convolucional Densamente Conectado (DenseNet) y Cápsulas matriciales con enrutamiento EM, en el cual se vio como es que estas están diseñadas y como trabajan.

3. Conclusión en base al segundo objetivo específico:

   a) Se diseño un modelo de red de cápsulas matriciales con enrutamiento Em utilizando una red convolucional densa. El uso de una red convolucional densa reduce significativamente el tiempo de entrenamiento y mejora la precisión. Esto es debido al procesamiento que tiene densenet, el cual concatena los mapas de características y no las reemplaza como es el caso de una red convolucional simple.

4. Conclusión en base al tercer objetivo específico:

   a) El entrenamiento del modelo base y propuesto se realizó con un total de 30 épocas. Obteniendo mejores resultados el modelo propuesto, siendo más optimo en tiempo de entrenamiento, precisión(accuracy) y pérdida de entrenamiento(training loss) frente al modelo base.

5. Conclusiones en base al cuarto objetivo específico:

a) Al utilizar una iteración de 1 se observa que el tiempo de entrenamiento es mucho menor que al utilizar una iteración de 2 y 3, sin embargo la precisión (accuracy) con iteración 1 es mucho menor que al utilizar una iteración de 2 y a la vez es superior a una iteración de 3, esto mismo ocurre con la perdida de entrenmiento(training loss). Por ende se concluye que la mejor opción es una iteración de 2 cuyo resultado es de 94.27 % para precisión(accuracy), 0.72 % de pérdida de entrenamiento(training loss) en un tiempo de 2 horas y 8 minutos.

b) Se observa que la mejor opción es utilizar una tasa de crecimiento de 32 con una iteración de 2 en la capa convolucional densa ya que se obtiene un mejor resultado para la precisión(accuracy), 94.27 % y la pérdida de entrenamiento(training loss), 0.72 %.

c) Otra mejora que se observa fue en el hardware, se observa que en el modelo propuesto se utiliza un 40 % de rendimiento del GPU a diferencia del modelo base que utiliza de un 90 % del GPU.

d) Estos buenos resultados son debido al procesamiento que tiene densenet, el cual concatena los mapas de características y no las reemplaza como es el caso de una red convolucional simple. Concatenar los mapas de características mejora el flujo de información, alivian el gradiente de fuga, fortalecen la propagación de información y elimina datos redundantes, conllevando así a obtener un menor tiempo de entrenamiento, una alta precisión y una menor pérdida.

6. Así mismo, la propuesta de diseño de modelo inicialmente a sido presentada en calidad de paper en la 5° Jornada de Inteligencia Artificial en el marco de la 16th Ibero-American Conference on Artificial Intelligence(IBERAMIA'2018), habiendo sido validada y expuesta a la comunidad científica del área.(Anexo 1).

Recomendaciones y Trabajos Futuros

- Para mejores resultados se recomienda utilizar un GPU de mayor potencia(NVIDEA 2080) o GPU's que estén diseñadas para trabajos en Machine Learning(NVIDEA V100, NVIDEA K80).
- Se deja como trabajo a futuro construir un dataset en 3D para un caso mucha mas realista. Esto quiere decir construir un data set con las mismas características que el dataset SmallNORB pero orientado hacia temas como reconocimientos de rostros, símbolos, lenguaje de señas, estructuras, o en campos como la medicina, biología, etc.
- Se deja como trabajo a futuro mejorar las capas ConvCaps1 y ConvCaps2. Estas capas en su estructura utilizan redes convolucionales simple, se plantea como objetivo reemplazar estas redes convolucionales por redes convolucionales densas y medir cuan mejor es el modelo de Matriz de Capsula con enrutamiento EM aplicando estas mejoras.
- Se deja como trabajo a futuro emplear otro tipo de red convolucional, siendo más específicos ResNet. Dicha red es similar a Densenet con la diferencia de que en vez de concatenar los mapas de características esta lo que hace es sumarlas.

Referencias

Agatonovic-Kustrin, S., y Beresford, R. (2000). Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, *22*(5), 717–727.

Brain tumor type classification via capsule networks, author=Parnian Afshar, Arash Mohammadi, Konstantinos N. Plataniotis, year=2018. (s.f.).

Capsule networks against medical imaging data challenges, author=Amelia Jiménez Sánchez, Shadi Albarqouni, Diana Mateus, year=2019. (s.f.).

Capsule networks for brain tumor classification based on mri images and coarse tumor boundaries, author=Parnian Afshar, Konstantinos N. Plataniotis, Arash Mohammadi, journal=IEEE, year=2019. (s.f.).

Chauhan, A., Babu, M., Kandru, N., y Lokegaonkar, S. (2018). *Empirical study on convergence of capsule networks with various hyperparameters.*

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., y Schmidhuber, J. (2011). High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*.

Duarte, K., Rawat, Y., y Shah, M. (2018). Videocapsulenet: A simplified network for action detection. En S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, y R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 7610–7619). Curran Associates, Inc.

The expectation maximization algorithm, author=IEEE Signal Processing Magazine, year=2000. (s.f.).

Fernando Izaurieta, C. S. (2015). Redes neuronales artificiales.

Gao Huang, L. v. d. M. K. Q. W., Zhuang Liu. (2016). Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

Garre, M., Cuadrado, J. J., Sicilia, M. A., Rodríguez, D., y Rejas, R. (2007). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software. *REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software*, *3*(1), 6–22.

Geoffrey E Hinton, S. S., y Frosst, N. (2017). Dynamic routing between capsules. En I. Guyon y cols. (Eds.), *Advances in neural information processing systems 30* (pp. 3856–3866). Curran Associates, Inc.

Gong, J., Qiu, X., Wang, S., y Huang, X. (2018). Information aggregation via dynamic routing for sequence encoding. *arXiv preprint arXiv:1806.01501*.

Hinton, G. E., Sabour, S., y Frosst, N. (2018). Matrix capsules with EM routing. En *International conference on learning representations.*

Hui, J. (2017). Understanding matrix capsules with em routing. *Blog. Nov*.

Isasi Vinuela, P., y Galván León, I. (2004). Redes de neuronas artificiales. *Un Enfoque Práctico, Editorial Pearson Educación SA Madrid España*.

Izaurieta, F., y Saavedra, C. (2000). Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*.

Jaiswal, A., AbdAlmageed, W., Wu, Y., y Natarajan, P. (2018, September). Capsulegan: Generative adversarial capsule network. En *The european conference on computer vision (eccv) workshops.*

Kakillioglu, B., Ahmad, A., y Velipasalar, S. (2018, Nov). Object classification from 3d volumetric data with 3d capsule networks. En *2018 ieee global conference on signal and information processing (globalsip)* (p. 385-389). doi: 10.1109/GlobalSIP.2018.8646333

LaLonde, R., y Bagci, U. (2018). Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*.

Martín del Brío, B., y Sanz, A. (2002). Redes neuronales y sistemas difusos. *Edit. Alfa Omega Ra-Ma. Madrid. España*.

Mukhometzianov, R., y Carrillo, J. (2018). Capsnet comparative performance evaluation for image classification. *arXiv preprint arXiv:1805.11195*.

Pacheco, M. (2017). *Identificacion de sistemas no lineales con redes neuronales convolucionales.* Cuidad de Mexico: CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS.

Phaye, S. S. R., Sikka, A., Dhall, A., y Bathula, D. (2018). Dense and diverse capsule networks: Making the capsules learn better. *arXiv preprint arXiv:1805.04001*.

Rajasegaran, J., Jayasundara, V., Jayasekara, S., Jayasekara, H., Seneviratne, S., y Rodrigo, R. (2019, June). Deepcaps: Going deeper with capsule networks. En *The ieee conference on computer vision and pattern recognition (cvpr).*

Schuster, N., Chess, M., y Yang, T. (2017). Exploration of capsule networks.

Shahroudnejad, A., Afshar, P., Plataniotis, K. N., y Mohammadi, A. (2018). Improved explainability of capsule networks: Relevance path by agreement. En *2018 ieee global conference on signal and information processing (globalsip)* (pp. 549–553).

Wang, D., y Liu, Q. (2018). An optimization view on dynamic routing between capsules.

Xi, E., Bing, S., y Jin, Y. (2017). Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*.

Zhang, S., Zhou, Q., y Wu, X. (2018). Fast dynamic routing based on weighted kernel density estimation. En *International symposium on artificial intelligence and robotics* (pp. 301–309).

Zhao, W., Ye, J., Yang, M., Lei, Z., Zhang, S., y Zhao, Z. (2018). Investigating capsule networks with dynamic routing for text classification. *arXiv preprint arXiv:1804.00538*.

Apéndice A

Anexo I

Diseño del Modelo Hibrido de Matriz de Capsulas Con EM Routing y una Red Convolucional Densa: DENSE EM ROUTING

Paul Theo Bernal Ríos - Washington Blas Huaman - Yeshica Isela Ormeño Ayala
Universidad Nacional de San Antonio Abad del Cusco
{113549, 121958, *yeshica.ormeno*} @*unsaac.edu.pe*

Resumen

Las redes de cápsulas (CapsNet), al ser una nueva arquitectura de aprendizaje profundo, motiva entender su funcionamiento para su aplicabilidad en diversas investigaciones. La idea de CapsNet surge para mejorar la clasificación de imágenes cuando éstos presentan rotación, inclinación u otra orientación diferente, obteniendo información más precisa. Los modelos como: enrutamiento dinámico de cápsulas, matriz de cápsulas con EM Routing, CapsNets Densos (DCNET) y CapsNets Dispersos (DCNET++), han demostrado mejoras progresivas en cuanto a datos complejos, en comparación a modelos muy utilizados como las redes neuronales convolucionales (CNNs). Por tanto, en esta investigación se elabora un nuevo modelo de CapsNet basado en Matriz de Cápsulas con EM Routing, reemplazando su primera capa de red convolucional simple (ReLU Conv1) por una red convolucional densa (DenseNet), para mostrar una mejora en la extracción de características y mayor precisión frente al modelo base.

Matriz de Cápsulas, EM Routing, CapsNet, DenseNet, Modelo, Concytec, Cienciactiva

1. Introducción

El aprendizaje profundo fue ideado para una mejora del aprendizaje automático. Actualmente la arquitectura de aprendizaje profundo con mayor éxito y más utilizado en el reconocimiento y la clasificación de imágenes, visión computacional, conducción automática de vehículos, es la red neuronal convolucional (CNN).

La introducción de redes de cápsulas (CapsNet) por Hinton[4], como una nueva arquitectura de aprendizaje profundo está motivando a investigadores, entender como funciona CapsNet para aplicarlo a su propia investigación [5][7]. La idea de CapsNet se desarrolló originariamente para mejorar la clasificación de imágenes en 2D cuando éstos presentan rotación, inclinación u otra orientación diferente, facilitando la obtención de información para la verificación de sus posiciones relativas.

Teniendo en cuenta que las CNNs tienen limitaciones ya que no son robustas para las transformaciones afines, es decir, un ligero cambio en la posición de las CNNs de objetos modifican su predicción. Aunque este problema puede reducirse hasta cierto punto mediante el aumento de datos durante el entrenamiento, esto no hace que la red sea robusta para ninguna nueva postura o variación que pueda estar presente en los datos de prueba. Otro inconveniente importante es que las CNNs genéricas no tienen en cuenta las relaciones espaciales entre los objetos de una imagen al tomar una decisión [6]. Es decir, simplemente las CNNs solo usan la mera presencia de ciertos objetos locales en una imagen para tomar una decisión, mientras que en realidad el contexto espacial de los objetos presentes es igualmente importante.

Por tanto en este trabajo se planteará un nuevo modelo de CapsNet basado en EM Routing concatenado con una red convolucional densa (DenseNet) de 8 capas y verificar si la obtención de datos, procesamiento y evaluación mejoran el rendimiento a modelos ya propuestos tales como CapsNets Densos y Dispersos (DCNET y DCNET++) que trabajan sobre el modelo básico de cápsulas, matriz de cápsulas con EM-Routing y enrutamiento dinámico de cápsulas, con el fin de proveer mayor precisión en los datos.

2. Antecedentes

Un avance incremental en el intento de abordar las limitaciones de representación de CNN como son la identificación de la posición y rotación, provino de Hinton quien introdujo el concepto de çápsulas". Las cápsulas abordaron estas ineficiencias representacionales de las CNNs con matrices de transformación, lo que permite a las redes aprender automáticamente relaciones entre partes y generalizar nuevos puntos de vista, algo en lo que las CNN tradicionales muestran un rendimiento exponencialmente bajo.
Con el objetivo de mejorar su modelo Hinton [4] describió una versión de cacápsulapsula en la que cada cápsula tiene una unidad logistica para representar la presencia de una entidad y una matriz 4x4 que aprende a representar la relación entre esa entidad y el espectador (la pose).
Aprovechando las ventajas que ofrecía las capsulas se plantearon diferentes modelos:
Kai Qiao [6] y su equipo teniendo en mente que la reconstrucción precisa de los estímulos de la imagen de la FMRI humana es un trabajo todavía desafiante, se plantearon como objetivo la reconstrucción visual precisa y para ello propone un nuevo método de reconstrucción visual basado en la arquitectura de una red de cápsulas (CapsNet), al cual denominaron CNAVR y llegando a la conclusión de que este modelo propuesto concuerda mejor con la corteza visual humana cuando se trata de una reconstrucción visual.
Edgar Xi, Selina Bing y Yang Jin [2] con el objetivo encontrar un modelo basado en CapsNet, el cual produzca un error de prueba óptimo. Plantearon varios modelos que van desde apilar más capas de cápsula hasta probar diferentes parámetros, se proponen los siguientes enfoques: apilamiento de más capas de cápsulas, aumentar el número de cápsulas primarias, modificar la escala de perdida en la reconstrucción y utilizar una función de activación personalizada llegando a conseguir resultados prometedores.
Dai Quoc Nguyen, Thanh Vu, Tu Dinh Nguyen, Dinh Phung [1] buscando adaptar los resultados de búsqueda a cada usuario específico en función de los intereses y preferencias personales del usuario es por ello que plantearon un modelo de inclusion basado en CapsNet para modelar las relaciones tripartitas para la personalización de búsquedas encontrando un resultado óptimo.

3. Propuesta del modelo

La propuesta de modelo esta esquematizada en la siguiente Figura 1 que está basado en el Modelo propuesto con una DenseNet de 8 capas, 32 cápsulas primarias con EM Routing, 2 capas de cápsulas convolucionales con EM Routing y finalmente 10 clases de cápsulas, todo esto efectuado en N epocas.

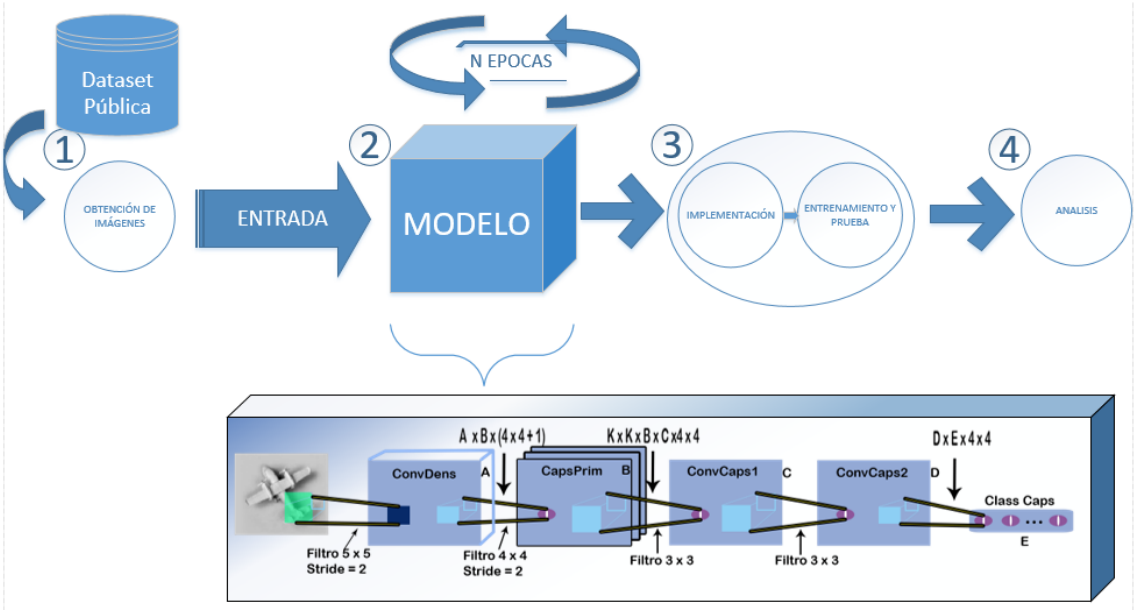

Figura 1: Modelo Hibrido de Matriz de Cápsulas con Em Routing y una Red Convolucional Densa.
Fuente: Creación propia

La primera acción corresponde a la obtención de imágenes. Esto se efectuará descargando datasets públicas de imágenes y/o fotografiando imágenes que podrían ser de plantas medicinales, (Paso 1).
Como segunda acción estando las imágenes en el dataset se inicia la construcción del modelo, que consiste en la agregación de una capa de red convolucional densa (DenseNet) de 8 capas con filtros de 5x5 y stride de 2 que da salida a 32 mapas de características que se conectan con las cápsulas primarias. En las cápsulas primarias, aplicamos un filtro de convolución 1x1 para transformar cada uno de los 32 mapas de características en 32 cápsulas primarias (B). Cada cápsula contiene una matriz de pose 4x4 (para obtener las posición y orientación relativa de cada objeto). Usamos la capa de convolución regular para implementar cápsulas primarias, agrupamos A x B x (4x4 + 1) neuronas para generar 32 cápsulas primarias. Las cápsulas primarias van seguidas por una capa de cápsula de convolución ConvCaps1 que usa 3x3 filtros(K) con un stride de 2. ConvCaps1 toma a las cápsulas como entrada y salida, además es similar a una capa de convolución normal, excepto que utiliza el enrutamiento EM para calcular la salida de la cápsula. La salida de la cápsula de ConvCaps1 se alimenta a ConvCaps2 que es otra capa de cápsula de convolución pero con un stride de 1. Las cápsulas de salida de ConvCaps2 están conectadas a las Class Caps con un filtro de 1x1 el cual genera una cápsula por clase, (Paso 2).
Como tercera acción se efectuaría la implementación del modelo utilizando Python y Tensorflow. Una vez implementado el modelo se pasa al entrenamiento de las imágenes obtenidas en N epocas. Luego se pasa a la prueba del modelo con imágenes de prueba, también obtenidas en la primera acción, (Paso 3). Finalmente como última acción se verificaría y validaría la prueba del modelo utilizando imágenes que podrían ser de las plantas que medicinales de la region del Cusco (Paso 4).

4. Conclusión y trabajos futuros

Con el modelo que se propone, se pretende facilitar la obtención de información para la verificación y validación de imágenes. Este modelo coadyuva a validar información y datos que son requeridos en diversas investigaciones o aplicables a proyectos de investigación. El aprendizaje profundo coadyuva a diversos procesos de desarrollo software que puedan incluir análisis y validación de información cuando éstos son altamente complejos. En vista a aplicar este modelo se plantean acciones futuras como:

- Obtención de imágenes a partir de datasets ya existentes (públicas) o elicitadas previamente.
- Implementación del modelo propuesto mediante un lenguaje de programación.
- Entrenamiento y pruebas del modelo.
- Análisis de resultados a partir de la prueba del modelo que valide la información procesada.

Al ser un modelo, su aplicación está orientada para investigaciones que requieran precisión y clasificación de la información (imágenes) que se tiene o se pueda obtener.

5. Agradecimientos

Este trabajo esta financiado por el Programa Yachaininchis Wiñariñampac Fondecyt Unsaac mediante el Proyecto de Investigación con fondos Canon “Aplicación de la Ingeniería de Requisitos a los proyectos de Investigación de la UNSAAC en desarrollo de Software y uso de TICs”

Referencias

[1] Dai Quoc Nguyen, T. V., Tu Dinh Nguyen, Dinh Phung. A Capsule Network-based Embedding Model for Search Personalization, Australia, 2018.

[2] Edgar Xi, S. B., Yang Jin. Capsule Network Performance on Complex Data, Pittsburgh, 2017.

[3] Gao Huang, Z. L., Kilian Q.Weinberger. Densely Connected Convolutional Networks IEEE, Honolulu, HI, USA ,2017.

[4] Geoffrey E. Hinton, S. S., Nicholas Frosst. Dynamic routing between capsules,Toronto, Canada, 2017.

[5] Geoffrey Hinton, S. S., Nicholas Frosst. Matrix Capsules With Em Routing, Toronto, Canada, 2018.

[6] Kai Qiao, C. Z., Linyuan Wang, Bin Yan, Jian Chen, Lei Zeng, Li, Tong. Accurate reconstruction of image stimuli from human fMRI based on the decoding model with capsule network architecture, Zhengzhou, China, 2018.

[7] Sai Samarth R Phaye, A. S., Abhinav Dhall, Deepti, Bathula. Dense and Diverse Capsule Networks: Making the Capsules Learn Better, Punjab, India, 2018.