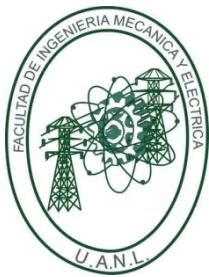




**Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica**



INTELIGENCIA ARTIFICIAL

PIA: ENTRENAMIENTO DE UNA RED NEURONAL CONVOLUCIONAL

1723100 Cindy Jannet González Leal

1801940 Ana Camila Ramírez García

1793880 Carlos Arturo Torres Cisneros

1742422 Luis Pablo Escobedo Tamez

1615458 Luis Enrique Palacios Nacianceno

Docente: Ing. Daniel Isaías López Páez

SEMESTRE: AGOSTO-DICIEMBRE 2021

Martes N4

San Nicolás de los Garza, N.L.

21 de noviembre del 2021

I. Introducción

En la última década las Redes Neuronales Artificiales (ANN) han recibido un interés particular como una tecnología para minería de datos, puesto que ofrece los medios para modelar de manera efectiva y eficiente problemas grandes y complejos.

Los modelos de ANN son dirigidos a partir de los datos, es decir, son capaces de encontrar relaciones (patrones) de forma inductiva por medio de los algoritmos de aprendizaje basado en los datos existentes más que requerir la ayuda de un modelador para especificar la forma funcional y sus interacciones.

Las ANN son un método de resolver problemas, de forma individual o combinadas con otros métodos, para aquellas tareas de clasificación, identificación, diagnóstico, optimización o predicción en las que el balance datos/conocimiento se inclina hacia los datos y donde, adicionalmente, puede haber la necesidad de aprendizaje en tiempo de ejecución y de cierta tolerancia a fallos.

En estos casos las RNAs se adaptan dinámicamente reajustando constantemente los “pesos” de sus interconexiones. Las ANN se basan en la analogía que existe en el comportamiento y función del cerebro humano, en particular del sistema nervioso, el cual está compuesto por redes de neuronas biológicas que poseen bajas capacidades de procesamiento, sin embargo, toda su capacidad cognitiva se sustenta en la conectividad de éstas.

II. Antecedentes

Entre las décadas de 1950 y 1960 el científico Frank Rosenblatt, inspirado en el trabajo de Warren McCulloch y Walter Pitts creó el Perceptron, la unidad desde donde nacería y se potenciarían las redes neuronales artificiales.

Un perceptron toma varias entradas binarias x_1, x_2, \dots etc y produce una sola salida binaria. Para calcular la salida, Rosenblatt introduce el concepto de “pesos” w_1, w_2, \dots , un número real que expresa la importancia de la respectiva entrada con la salida. La salida de la neurona será 1 o 0 si la suma de la multiplicación de pesos por entradas es mayor o menor a un determinado umbral.

Sus principales usos son decisiones binarias sencillas, o para crear funciones lógicas como OR, AND.

1965 – Multilayer Perceptron

Como se imaginarán, el multilayer perceptron es una “amplificación” del percepción de una única neurona a más de una. Además, aparece el concepto de capas de entrada, oculta y salida. Pero con valores de entrada y salida binarios. No olvidemos que tanto el valor de los pesos como el de umbral de cada neurona lo asignaba manualmente el científico. Cuantos más perceptrones en las capas, mucho más difícil conseguir los pesos para obtener salidas deseadas.

Neuronas Sigmoides

Para poder lograr que las redes de neuronas aprendieran solas fue necesario introducir un nuevo tipo de neuronas.

Las llamadas Neuronas Sigmoides son similares al perceptron, pero permiten que las entradas, en vez de ser ceros o unos, puedan tener valores reales como 0,5 ó 0,377 ó lo que sea.

También aparecen las neuronas “bias” que siempre suman 1 en las diversas capas para resolver ciertas situaciones. Ahora las salidas en vez de ser 0 ó 1, será $d(w \cdot x + b)$ donde d será la función sigmoide definida como $d(z) = 1/(1 + e^{-z})$. Esta es la primer función de activación!

Con esta nueva fórmula, se puede lograr que pequeñas alteraciones en valores de los pesos (deltas) producen pequeñas alteraciones en la salida. Por lo tanto, podemos ir ajustando muy de a poco los pesos de las conexiones e ir obteniendo las salidas deseadas.

Redes Feedforward

Se les llama así a las redes en que las salidas de una capa son utilizadas como entradas en la próxima capa. Esto quiere decir que no hay loops “hacia atrás”. Siempre se “alimenta” de valores hacia adelante. Hay redes que veremos más adelante en las que sí que existen esos loops (Recurrent Neural Networks).

Además, existe el concepto de “fully connected Feedforward Networks” y se refiere a que todas las neuronas de entrada, están conectadas con todas las neuronas de la siguiente capa.

1986 – Backpropagation

Gracias al algoritmo de backpropagation se hizo posible entrenar redes neuronales de multiples capas de manera supervisada. Al calcular el error obtenido en la salida e ir propagando hacia las capas anteriores se van haciendo ajustes pequeños (minimizando costo) en cada iteración para lograr que la red aprenda consiguiendo que la red pueda -por ejemplo- clasificar las entradas correctamente.

1989 – Convolutional Neural Network

Las Convolutional Neural Networks son redes multilayered que toman su inspiración del cortex visual de los animales. Esta arquitectura es útil en varias aplicaciones, principalmente procesamiento de imágenes. La primera CNN fue creada por Yann LeCun y estaba enfocada en el reconocimiento de letras manuscritas.

La arquitectura consta de varias capas que implementan la extracción de características y luego clasificar. La imagen se divide en campos receptivos que alimentan una capa convolutional que extrae features de la imagen de entrada (Por ejemplo, detectar líneas verticales, vértices, etc). El siguiente paso es pooling que reduce la dimensionalidad de las features extraídas manteniendo la información más importante. Luego se hace una nueva convolución y otro pooling que alimenta una red feedforward multicapa. La salida final de la red es un grupo de nodos que clasifican el resultado, por ejemplo, un nodo para cada número del 0 al 9 (es decir, 10 nodos, se “activan” de a uno).

III. Metodología

Para la realización de los 3 modelos usaremos : **Google Collab, Tensorflow y Keras**, programando en el lenguaje **Python**

Keras es una biblioteca de redes neuronales artificiales de código abierto. Está desarrollada en Python y puede ejecutarse sobre diferentes plataformas como TensorFlow.

Creado por el equipo de Google Brain, **TensorFlow** es una biblioteca de código abierto para la computación numérica y Machine Learning a gran escala. TensorFlow reúne una serie de modelos y algoritmos de Machine Learning y Deep Learning y los hace útiles mediante una metáfora común.

Google Collab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV. ... Todo ello con Python 2.7 y 3.6.

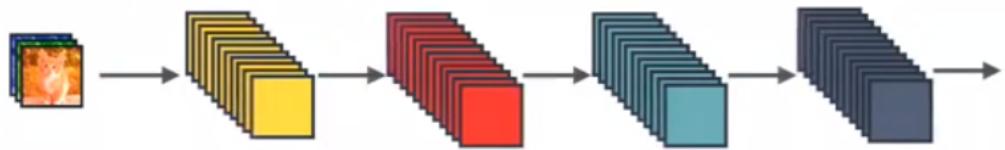
El data set utilizado: **CIFAR100**

Este conjunto de datos es como el CIFAR-10, excepto que tiene 100 clases que contienen 600 imágenes cada una. Hay 500 imágenes de entrenamiento y 100 imágenes de prueba por clase. Las 100 clases del CIFAR-100 se agrupan en 20 superclases. Cada imagen viene con una etiqueta "fina" (la clase a la que pertenece) y una etiqueta "gruesa" (la superclass a la que pertenece).

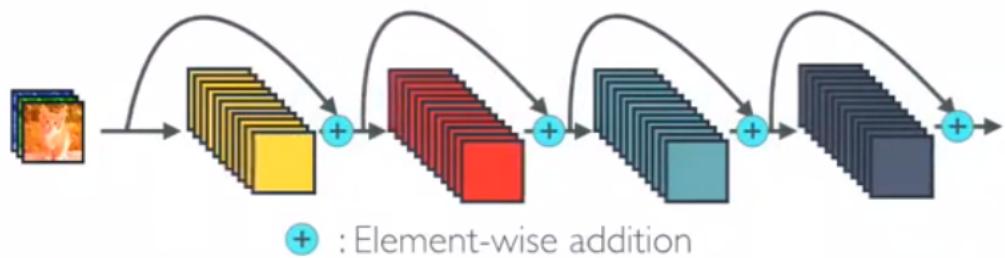
Superclases de Cifar100:

aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

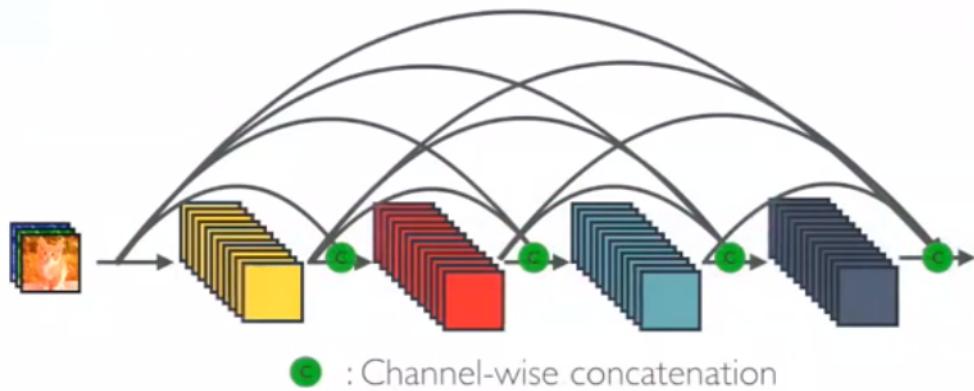
Ahora bien, una DenseNet es un tipo de red neuronal convolucional que utiliza conexiones densas entre capas, a través de bloques densos, donde conectamos todas las capas (con tamaños de mapas de características coincidentes) directamente entre sí. Para preservar la naturaleza de retroalimentación, cada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



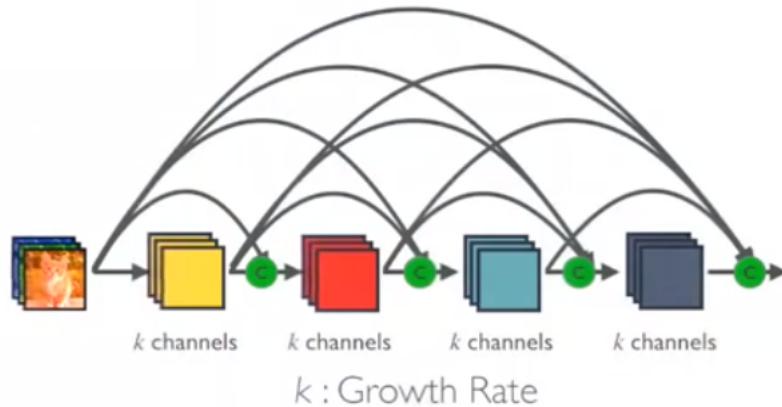
En Standard ConvNet, la imagen de entrada pasa por múltiples convoluciones y obtiene características de alto nivel.



En ResNet, se propone el mapeo de identidad para promover la propagación del gradiente. Se utiliza la adición de elementos. Puede verse como algoritmos con un estado que se pasa de un módulo ResNet a otro.

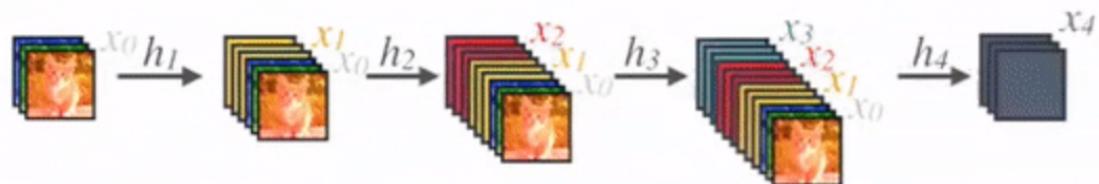


En DenseNet, cada capa obtiene entradas adicionales de todas las capas anteriores y pasa sus propios mapas de características a todas las capas posteriores. Se utiliza la concatenación. Cada capa está recibiendo un "conocimiento colectivo" de todas las capas anteriores.



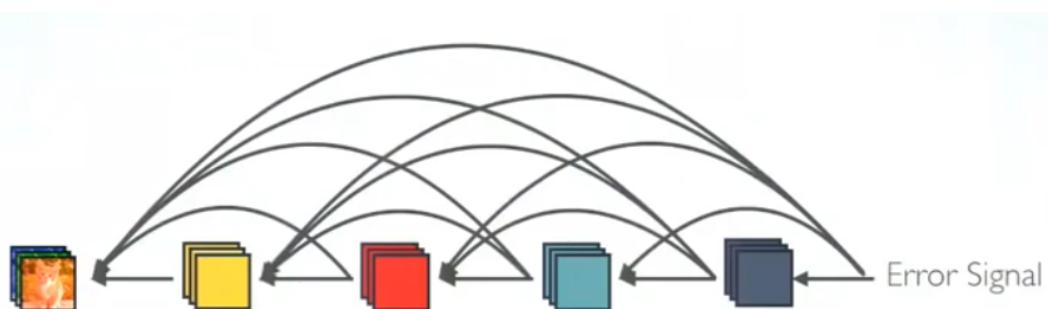
Dado que cada capa recibe mapas de características de todas las capas anteriores, la red puede ser más delgada y compacta, es decir, el número de canales puede ser menor. La tasa de crecimiento k es el número adicional de canales para cada capa.

Por lo tanto, tiene una mayor eficiencia computacional y eficiencia de memoria. La siguiente figura muestra el concepto de concatenación durante la propagación hacia adelante:

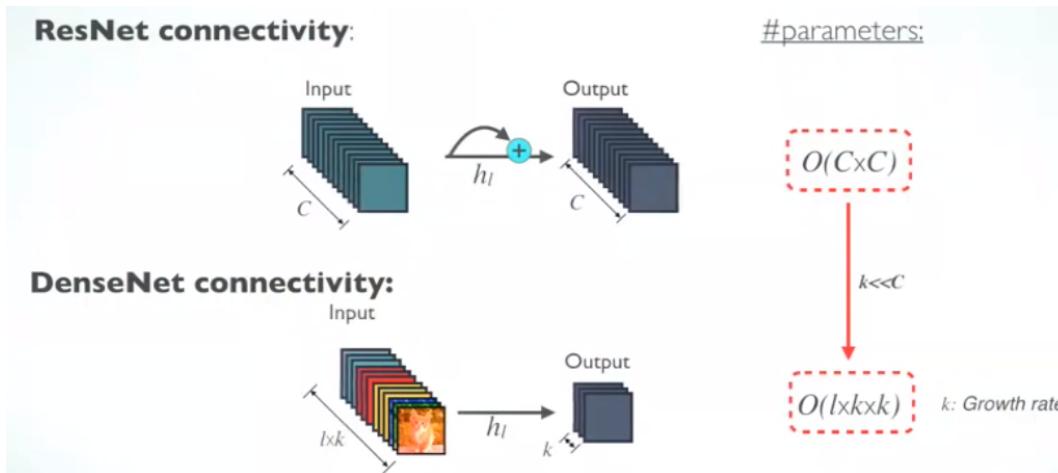


Con respecto a las ventajas de la DenseNet se tienen las siguientes:

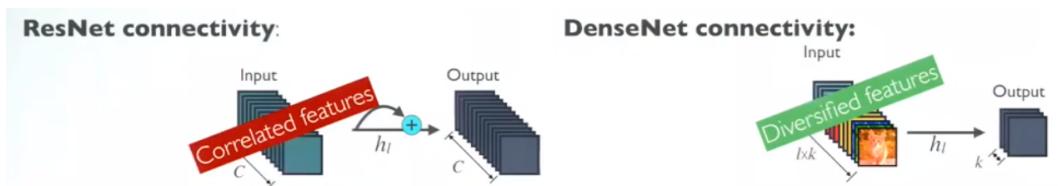
- Fuerte flujo de gradiente: la señal de error se puede propagar fácilmente a capas anteriores de forma más directa. Este es un tipo de supervisión profunda implícita, ya que las capas anteriores pueden obtener supervisión directa de la capa de clasificación final.



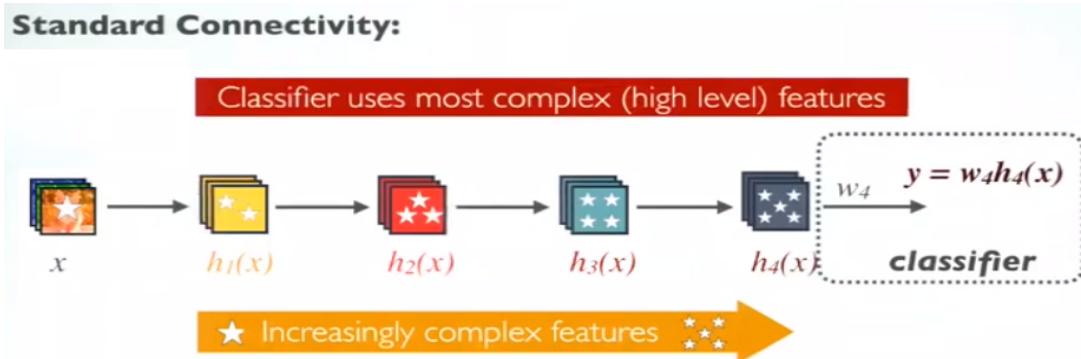
- b. Parámetro y eficiencia computacional: Para cada capa, el número de parámetros en ResNet es directamente proporcional a $C \times C$, mientras que el número de parámetros en DenseNet es directamente proporcional a $l \times k \times k$. Como $k \ll C$, DenseNet tiene un tamaño mucho más pequeño que ResNet.

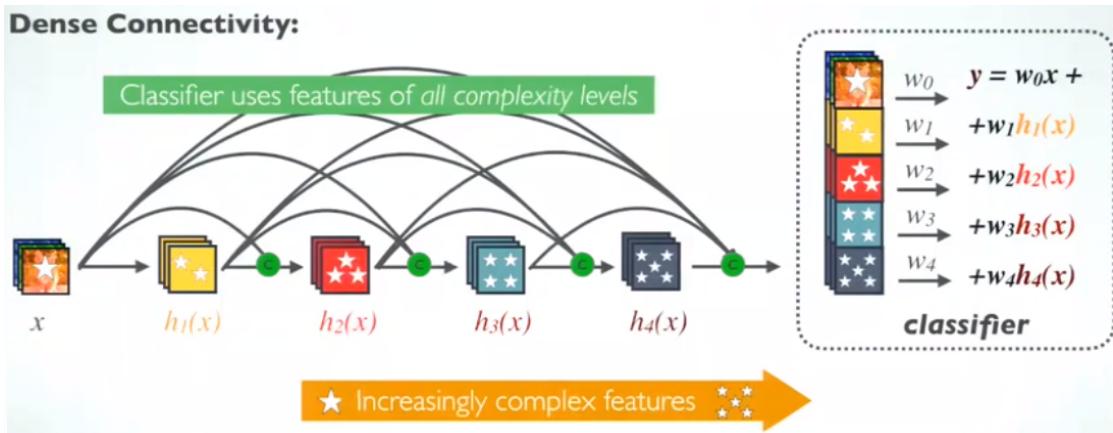


- c. Funciones más diversificadas: Dado que cada capa en DenseNet recibe todas las capas anteriores como entrada, características más diversificadas y tienden a tener patrones más ricos.

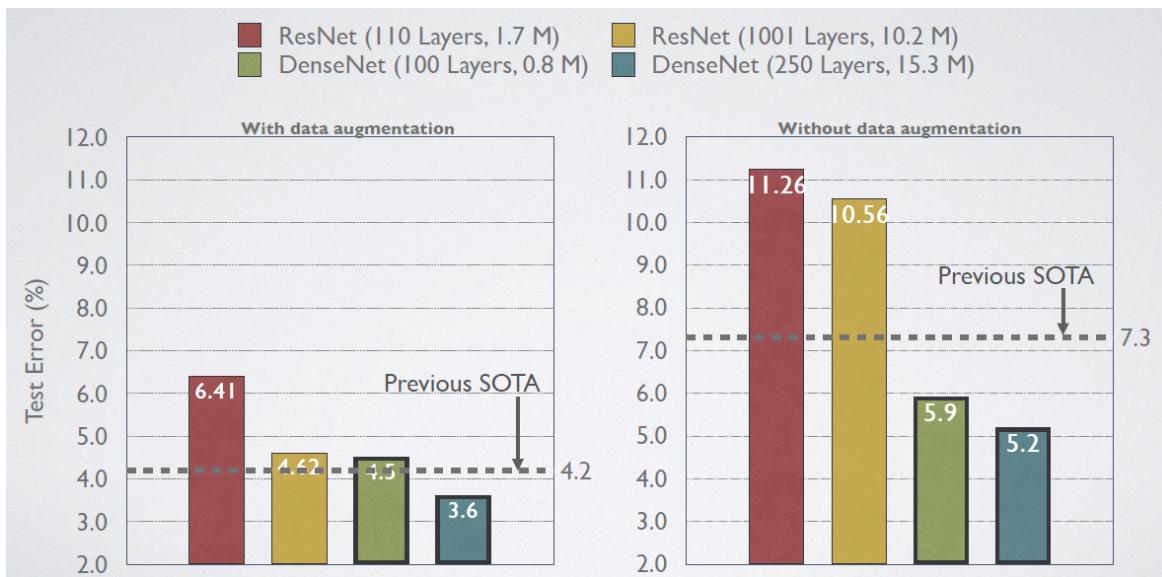


- d. Mantiene características de baja complejidad: En Standard ConvNet, el clasificador usa las características más complejas. En DenseNet, el clasificador utiliza características de todos los niveles de complejidad. Tiende a dar límites de decisión más fluidos. También explica por qué DenseNet funciona bien cuando los datos de entrenamiento son insuficientes.





Comparativa utilizando CIFAR-10 y CIFAR-100 con diferentes redes



Pre-Activación ResNet se utiliza en una comparación detallada.

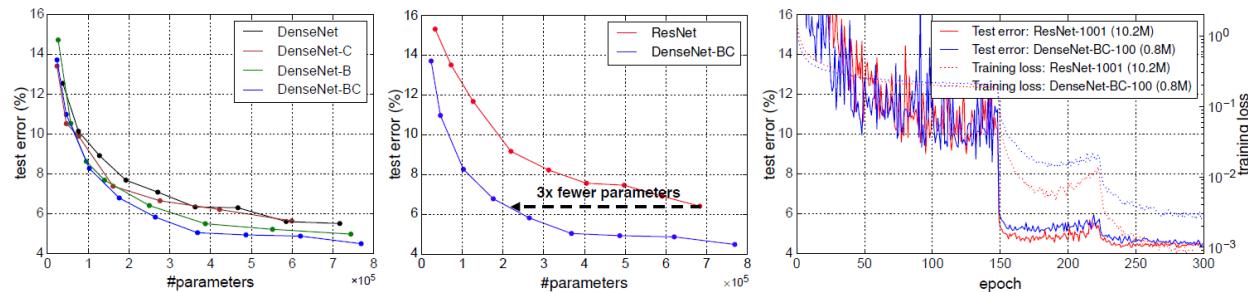
Con aumento de datos (C10 +), error de prueba:

- ResNet-110 de tamaño pequeño: 6,41%
- ResNet-1001 de gran tamaño (parámetros de 10,2 M): 4,62%
- Estado del arte (SOTA) 4,2%
- DenseNet-BC de tamaño pequeño ($L = 100, k = 12$) (solo 0,8 M de parámetros): 4,5%
- DenseNet de gran tamaño ($L = 250, k = 24$): 3,6%

Sin aumento de datos (C10), error de prueba:

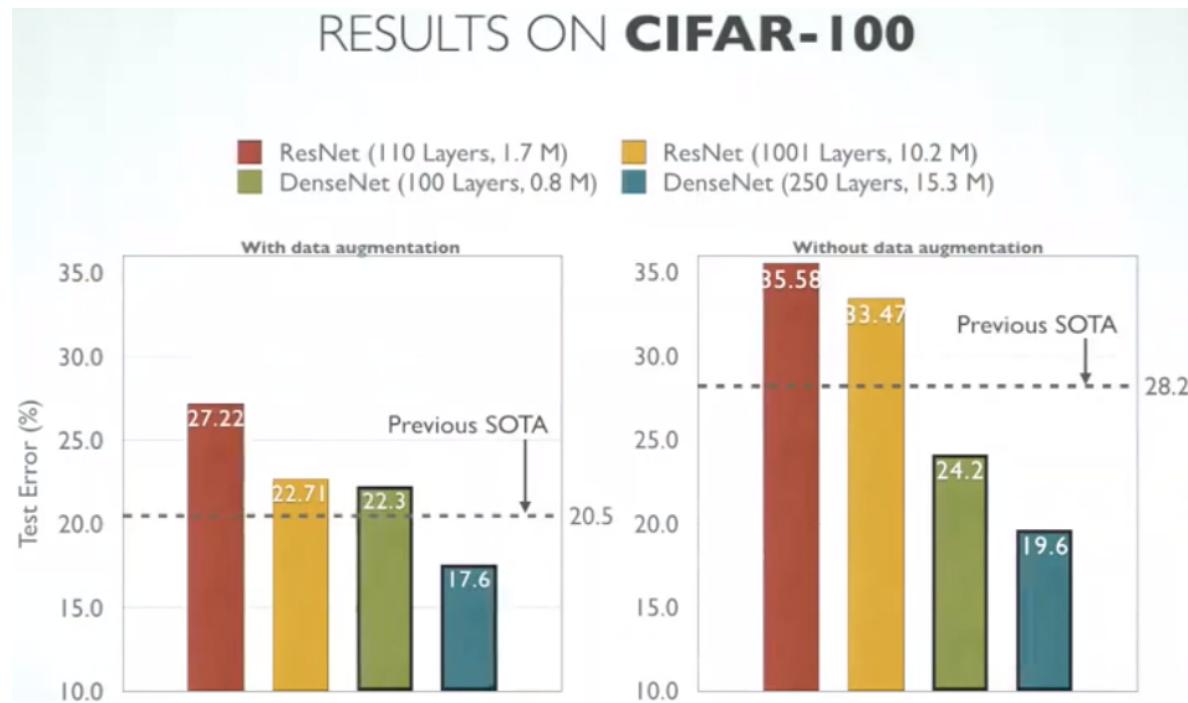
- ResNet-110 de tamaño pequeño: 11,26%
- ResNet-1001 de gran tamaño (parámetros de 10,2 M): 10,56%
- Estado del arte (SOTA) 7,3%
- DenseNet-BC de tamaño pequeño ($L = 100, k = 12$) (solo 0,8 M de parámetros): 5,9%
- DenseNet de gran tamaño ($L = 250, k = 24$): 4,2%

Aparece un sobreajuste severo en Pre-Activación ResNet, mientras que DenseNet funciona bien cuando los datos de entrenamiento son insuficientes, ya que DenseNet usa características de todos los niveles de complejidad.



- Izquierda: DenseNet-BC obtiene los mejores resultados.
- Medio: Pre-Activación ResNet ya tiene menos parámetros en comparación con AlexNet y VGGNet, y DenseNet-BC ($k = 12$) obtuvo 3 veces menos parámetros que Pre-Activación ResNet con el mismo error de prueba.
- Derecha: DenseNet-BC-100 con 0.8 parámetros obtuvo un error de prueba similar al de Pre-Activación ResNet-1001 con 10.2M parámetros.

Tendencias similares en CIFAR-100 se presentan a continuación:



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17] with Dropout/Drop-path	21	38.6M	10.18	5.22	35.34	23.30	2.01
	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	7.00	5.24	27.55	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	5.77	4.10	23.79	20.20	1.67
DenseNet ($k = 24$)	100	27.2M	5.83	3.74	23.42	19.25	1.59
DenseNet-BC ($k = 12$)	100	0.8M	5.92	4.51	24.15	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	5.19	3.62	19.64	17.60	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	3.46	-	17.18	-

SVHN es el conjunto de datos de Street View House Numbers. El color azul significa el mejor resultado. DenseNet-BC no puede obtener un mejor resultado que el DenseNet básico, los autores argumentan que SVHN es una tarea relativamente fácil, y los modelos extremadamente profundos pueden sobreajustarse al conjunto de entrenamiento.

IV. Experimentación

En la plataforma Google Colab, utilizando la RAM, Disco Duro y el Runtime mediante la GPU, se logró realizar la experimentación. Comenzamos creando un modelo secuencial para la predicción, con optimizadores convolucional 2d, y max pool 2d, con los filtros kernel_size tomando en cuenta el tamaño 2d, el tamaño por imagen del dataset, con 3 capas densas de 256,128 y 100 con activaciones tipo Relu, y terminando con la activación softmax con un total de 1,103,300 parámetros en total.

```

↳ Model: "sequential_4"
=====
Layer (type)          Output Shape       Param #
conv2d_8 (Conv2D)     (None, 16, 16, 32)    416
max_pooling2d_5 (MaxPooling) (None, 16, 16, 32)  0
                                         (2D)
conv2d_9 (Conv2D)     (None, 8, 8, 64)      8256
max_pooling2d_6 (MaxPooling) (None, 8, 8, 64)  0
                                         (2D)
flatten_1 (Flatten)   (None, 4096)        0
dense_3 (Dense)       (None, 256)         1048832
dense_4 (Dense)       (None, 128)         32896
dense_5 (Dense)       (None, 100)         12900
=====
Total params: 1,103,300
Trainable params: 1,103,300
Non-trainable params: 0

```

```
Epoch 12/25
1563/1563 [=====] - 49s 31ms/step - loss: 0.5481 - accuracy: 0.8267
Epoch 13/25
1563/1563 [=====] - 49s 31ms/step - loss: 0.4749 - accuracy: 0.8484
Epoch 14/25
1563/1563 [=====] - 48s 31ms/step - loss: 0.4313 - accuracy: 0.8623
Epoch 15/25
1563/1563 [=====] - 48s 31ms/step - loss: 0.3909 - accuracy: 0.8745
Epoch 16/25
1563/1563 [=====] - 49s 31ms/step - loss: 0.3664 - accuracy: 0.8819
Epoch 17/25
1563/1563 [=====] - 49s 32ms/step - loss: 0.3409 - accuracy: 0.8912
Epoch 18/25
1563/1563 [=====] - 48s 31ms/step - loss: 0.3034 - accuracy: 0.9027
Epoch 19/25
1563/1563 [=====] - 48s 31ms/step - loss: 0.3118 - accuracy: 0.9006
Epoch 20/25
1563/1563 [=====] - 48s 31ms/step - loss: 0.2833 - accuracy: 0.9098
Epoch 21/25
1563/1563 [=====] - 49s 32ms/step - loss: 0.2756 - accuracy: 0.9131
Epoch 22/25
1563/1563 [=====] - 50s 32ms/step - loss: 0.2721 - accuracy: 0.9150
Epoch 23/25
1563/1563 [=====] - 50s 32ms/step - loss: 0.2656 - accuracy: 0.9167
Epoch 24/25
1563/1563 [=====] - 51s 32ms/step - loss: 0.2550 - accuracy: 0.9214
Epoch 25/25
1563/1563 [=====] - 50s 32ms/step - loss: 0.2476 - accuracy: 0.9231
<keras.callbacks.History at 0x7f1c6f130f10>
```

Tomando en cuenta 25 Epochs, Batch_Size=Default, obtenemos un entrenamiento alto, pero una efectividad baja, esto quiere decir que el mundo real el modelo no detectaría cambios en las diferentes imágenes que le fuesen presentadas.

SIGUIENTE MÉTODO

Dentro de la documentación de Keras y Tensorflow, nos brindan la opción de utilizar modelos re-entrenados que han sido probados con gran exactitud en proyectos de redes neuronales que son enfocados en la clasificación de imágenes.

Al realizar pruebas con la mayoría de los 28 modelos y distintos benchmarks obtuvimos como peor programa entrenado EfficientNet y como el mejor modelo que se adapta a nuestro dataset de Cifar100, la arquitectura de DenseNet, por lo que los modelos a continuación están basados en ello:

```

g data from https://storage.googleapis.com/keras-applications/efficientnetb0\_notop.h5
==] - 0s 0us/step
==] - 0s 0us/step

7s 175ms/step - loss: 4.6460 - accuracy: 0.0106 - val_loss: 4.6394 - val_accuracy: 0.0097
9s 173ms/step - loss: 4.6461 - accuracy: 0.0101 - val_loss: 4.6417 - val_accuracy: 0.0099
8s 172ms/step - loss: 4.6456 - accuracy: 0.0101 - val_loss: 4.6423 - val_accuracy: 0.0091
8s 172ms/step - loss: 4.6459 - accuracy: 0.0110 - val_loss: 4.6421 - val_accuracy: 0.0091
9s 173ms/step - loss: 4.6461 - accuracy: 0.0096 - val_loss: 4.6465 - val_accuracy: 0.0091
]Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb1\_notop.h5
==] - 0s 0us/step
==] - 0s 0us/step

3s 241ms/step - loss: 4.6611 - accuracy: 0.0105 - val_loss: 4.6488 - val_accuracy: 0.0097
0s 238ms/step - loss: 4.6592 - accuracy: 0.0099 - val_loss: 4.6528 - val_accuracy: 0.0110
0s 238ms/step - loss: 4.6610 - accuracy: 0.0104 - val_loss: 4.6531 - val_accuracy: 0.0091
8s 246ms/step - loss: 4.6609 - accuracy: 0.0095 - val_loss: 4.6570 - val_accuracy: 0.0096
1s 239ms/step - loss: 4.6598 - accuracy: 0.0106 - val_loss: 4.6572 - val_accuracy: 0.0086
]Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb2\_notop.h5
==] - 0s 0us/step
==] - 0s 0us/step

9s 255ms/step - loss: 4.6778 - accuracy: 0.0099 - val_loss: 4.6747 - val_accuracy: 0.0107
6s 252ms/step - loss: 4.6770 - accuracy: 0.0108 - val_loss: 4.6820 - val_accuracy: 0.0097
6s 252ms/step - loss: 4.6780 - accuracy: 0.0110 - val_loss: 4.6827 - val_accuracy: 0.0104
5s 252ms/step - loss: 4.6772 - accuracy: 0.0103 - val_loss: 4.6776 - val_accuracy: 0.0091

```

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
DenseNet121	33	0.750	0.923	8,062,504	121	77.14	5.38
DenseNet169	57	0.762	0.932	14,307,880	169	96.40	6.28
DenseNet201	80	0.773	0.936	20,242,984	201	127.24	6.67
NASNetMobile	23	0.744	0.919	5,326,716	-	27.04	6.70
NASNetLarge	343	0.825	0.960	88,949,818	-	344.51	19.96
EfficientNetB0	29	-	-	5,330,571	-	46.00	4.91
EfficientNetB1	31	-	-	7,856,239	-	60.20	5.55
EfficientNetB2	36	-	-	9,177,569	-	80.79	6.50
EfficientNetB3	48	-	-	12,320,535	-	139.97	8.77
EfficientNetB4	75	-	-	19,466,823	-	308.33	15.12
EfficientNetB5	118	-	-	30,562,527	-	579.18	25.29
EfficientNetB6	166	-	-	43,265,143	-	958.12	40.45
EfficientNetB7	256	-	-	66,658,687	-	1578.90	61.62

<https://keras.io/api/applications/>

Construcción de los Modelos: se enfocó en el estudio de [[AgniData](#)], donde brinda la oportunidad de conocer la lista de modelos y como aplicarlos correctamente.

Carpeta GitHub

Modelo 1:

Para la construcción del modelo 1, el modelo **DenseNet201** con 20,242,984 parámetros, y con los hiperparámetros de 10 épocas, un batch_size de 100, y una redimensión del 70% del data

LINK de la notebook en Google Collab:
https://github.com/carlosarturotorres/InteligenciaArtificialyRedesNeuronales/blob/main/LasActividadesVanAqui/PIA-Red%20Neuronal%20Convolucional/Modelo_1.ipynb

Modelo 2:

Para la construcción del modelo 2, usando **DenseNet169** y **DenseNet201** con 20,242,984 y 14,307,880 parámetros respectivamente, con los hiperparámetros de 3 épocas, un batch_size de 32, y una redimensión del 70% del data

LINK de la notebook en Google Collab:
https://github.com/carlosarturotorres/InteligenciaArtificialyRedesNeuronales/blob/main/LasActividadesVanAqui/PIA-Red%20Neuronal%20Convolucional/Modelo_2.ipynb

Modelo 3:

Para la construcción del modelo 3, el modelo **DenseNet201** con 20,242,984 parámetros, y con los hiperparámetros de 20 épocas, un batch_size de 100, y una redimensión del 70% del data

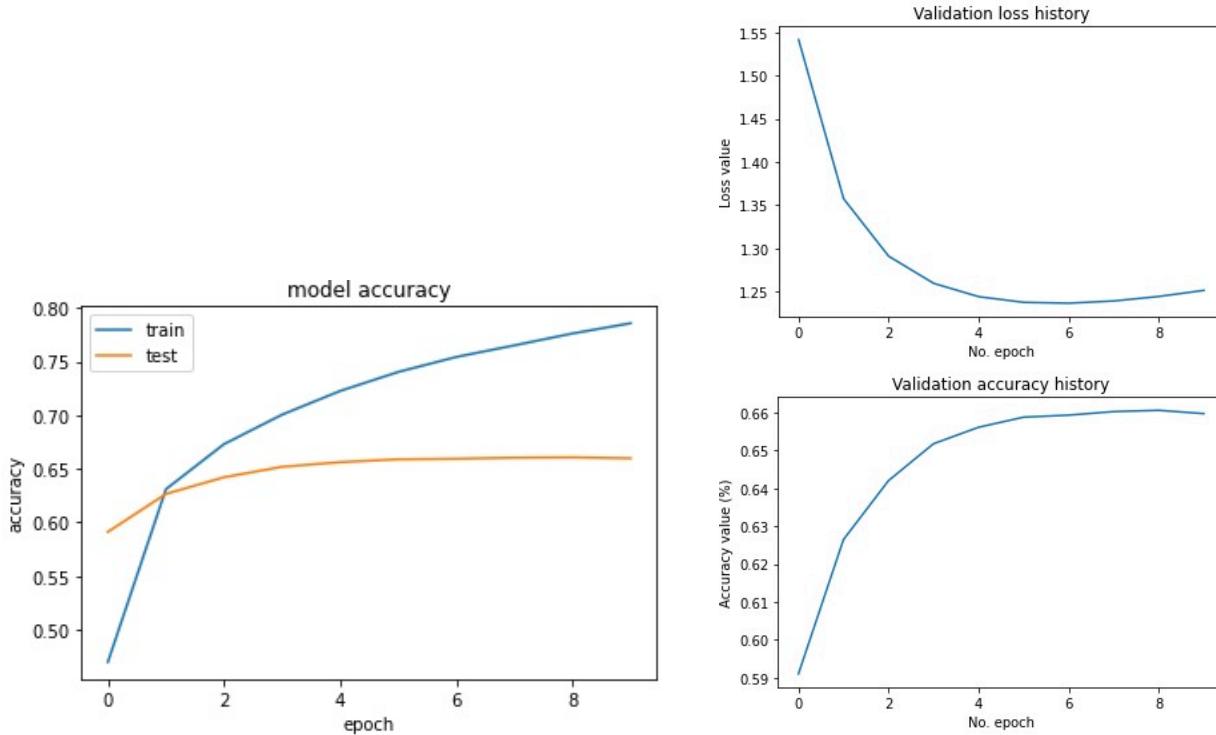
LINK de la notebook en Google Collab:
https://github.com/carlosarturotorres/InteligenciaArtificialyRedesNeuronales/blob/main/LasActividadesVanAqui/PIA-Red%20Neuronal%20Convolucional/Modelo_3.ipynb

V. Resultados

Modelo 1

```
0% | 0/1 [00:00<?, ?it/s] Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74842112/74836368 [=====] - 1s 0us/step
7485304/74836368 [=====] - 1s 0us/step
Epoch 1/10
350/350 [=====] - 465s 1s/step - loss: 2.1815 - accuracy: 0.4697 - val_loss: 1.5418 - val_accuracy: 0.5911
Epoch 2/10
350/350 [=====] - 416s 1s/step - loss: 1.3617 - accuracy: 0.6309 - val_loss: 1.3574 - val_accuracy: 0.6265
Epoch 3/10
350/350 [=====] - 416s 1s/step - loss: 1.1781 - accuracy: 0.6729 - val_loss: 1.2909 - val_accuracy: 0.6428
Epoch 4/10
350/350 [=====] - 413s 1s/step - loss: 1.0708 - accuracy: 0.7004 - val_loss: 1.2594 - val_accuracy: 0.6517
Epoch 5/10
350/350 [=====] - 413s 1s/step - loss: 0.9943 - accuracy: 0.7225 - val_loss: 1.2440 - val_accuracy: 0.6561
Epoch 6/10
350/350 [=====] - 413s 1s/step - loss: 0.9348 - accuracy: 0.7402 - val_loss: 1.2374 - val_accuracy: 0.6587
Epoch 7/10
350/350 [=====] - 414s 1s/step - loss: 0.8861 - accuracy: 0.7541 - val_loss: 1.2364 - val_accuracy: 0.6593
Epoch 8/10
350/350 [=====] - 414s 1s/step - loss: 0.8449 - accuracy: 0.7650 - val_loss: 1.2390 - val_accuracy: 0.6602
Epoch 9/10
350/350 [=====] - 415s 1s/step - loss: 0.8094 - accuracy: 0.7760 - val_loss: 1.2442 - val_accuracy: 0.6605
Epoch 10/10
350/350 [=====] - 416s 1s/step - loss: 0.7782 - accuracy: 0.7856 - val_loss: 1.2513 - val_accuracy: 0.6597
100% [=====] 1/1 [1:10:42<00:00, 4242.64s/it]
```

Valores obtenidos con el modelo 1 Epoca 10: Accuracy 78.56% y un Valor de Accuracy de 65.97%

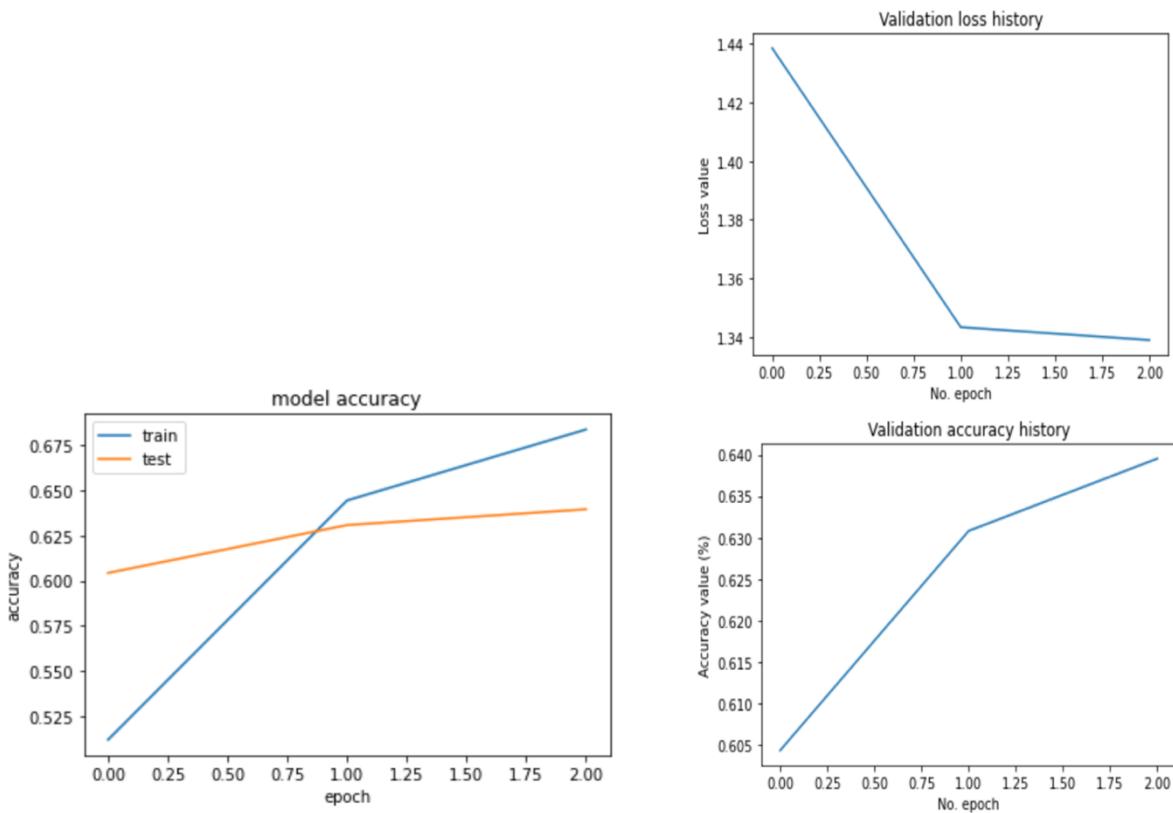


En las gráficas obtenidas vemos como despues de la epoca 2, empiezan a separarse el entrenamiento del test en nuestra red, en este caso se presenta un mayor overfitting entre ambos valores.

Modelo 2

```
0% | 0/2 [00:00<?, ?it/s] Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet169\_weights\_tf\_dim\_ordering\_tf\_kernels.h5
51879936/51877672 [=====] - 1s 0us/step
51888128/51877672 [=====] - 1s 0us/step
Epoch 1/3
1093/1093 [=====] - 468s 391ms/step - loss: 2.0103 - accuracy: 0.4858 - val_loss: 1.5299 - val_accuracy: 0.5843
Epoch 2/3
1093/1093 [=====] - 421s 386ms/step - loss: 1.3585 - accuracy: 0.6251 - val_loss: 1.4505 - val_accuracy: 0.6065
Epoch 3/3
1093/1093 [=====] - 421s 386ms/step - loss: 1.2175 - accuracy: 0.6663 - val_loss: 1.4465 - val_accuracy: 0.6132
50% | 1/2 [22:40<22:40, 1360.28s/it] Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201\_weights\_tf\_dim\_ordering\_tf\_kernels.h5
74842112/74836368 [=====] - 1s 0us/step
74850304/74836368 [=====] - 1s 0us/step
Epoch 1/3
1093/1093 [=====] - 551s 492ms/step - loss: 1.8956 - accuracy: 0.5119 - val_loss: 1.4385 - val_accuracy: 0.6043
Epoch 2/3
1093/1093 [=====] - 532s 487ms/step - loss: 1.2730 - accuracy: 0.6444 - val_loss: 1.3434 - val_accuracy: 0.6308
Epoch 3/3
1093/1093 [=====] - 533s 488ms/step - loss: 1.1377 - accuracy: 0.6838 - val_loss: 1.3390 - val_accuracy: 0.6395
100% | 2/2 [49:44<00:00, 1492.33s/it]
```

Valores obtenidos con el modelo 2 Epoca 3 Densenet169 : Accuracy 66.63% y un Valor de Accuracy de 61.32%,
 Epoca 3 Densenet201 Accuracy de 68.38% y Valor de Accuracy 63.95%



En las siguientes gráficas podemos observar una mayor cercanía entre los valores del entrenamiento con la exactitud, no es tan notorio el overfitting

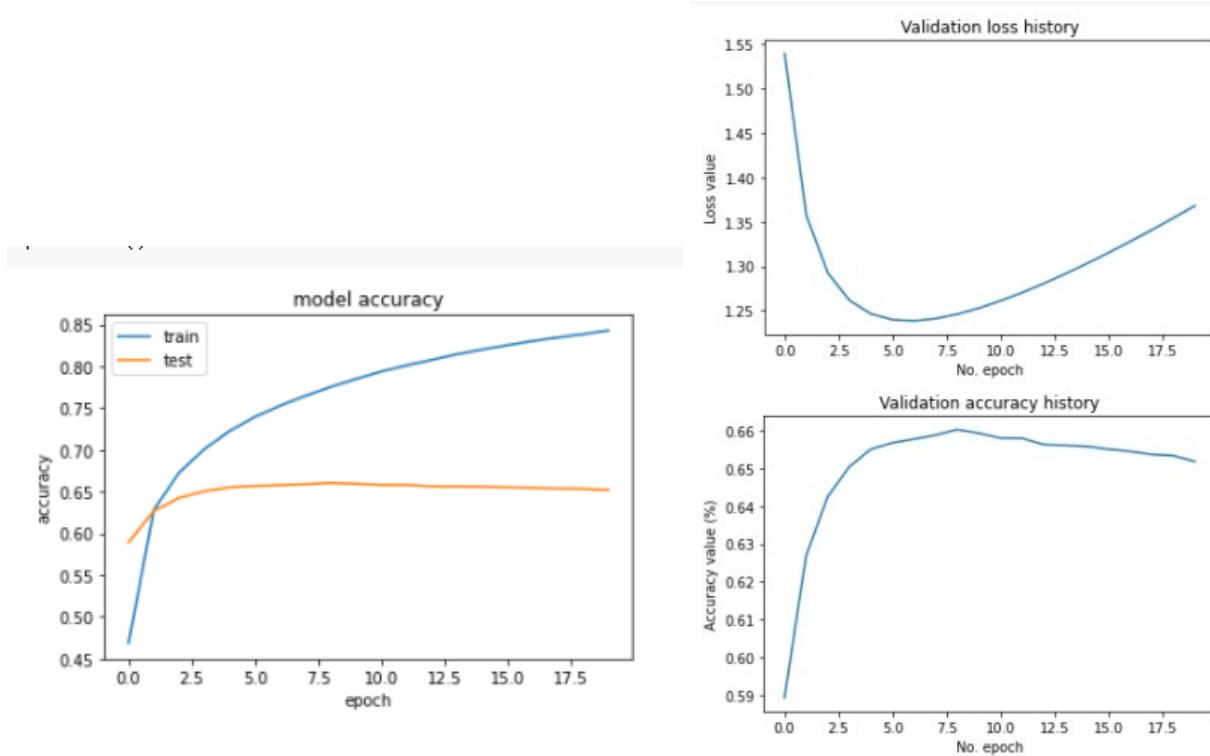
Modelo 3

```

Epoch 5/20
350/350 [=====] - 445s 1s/step - loss: 0.9949 - accuracy: 0.7227 - val_loss: 1.2464 - val_accuracy: 0.6551
Epoch 6/20
350/350 [=====] - 445s 1s/step - loss: 0.9354 - accuracy: 0.7395 - val_loss: 1.2398 - val_accuracy: 0.6568
Epoch 7/20
350/350 [=====] - 445s 1s/step - loss: 0.8867 - accuracy: 0.7529 - val_loss: 1.2387 - val_accuracy: 0.6578
Epoch 8/20
350/350 [=====] - 445s 1s/step - loss: 0.8456 - accuracy: 0.7648 - val_loss: 1.2412 - val_accuracy: 0.6589
Epoch 9/20
350/350 [=====] - 445s 1s/step - loss: 0.8101 - accuracy: 0.7754 - val_loss: 1.2461 - val_accuracy: 0.6603
Epoch 10/20
350/350 [=====] - 445s 1s/step - loss: 0.7789 - accuracy: 0.7850 - val_loss: 1.2530 - val_accuracy: 0.6593
Epoch 11/20
350/350 [=====] - 445s 1s/step - loss: 0.7512 - accuracy: 0.7938 - val_loss: 1.2612 - val_accuracy: 0.6581
Epoch 12/20
350/350 [=====] - 445s 1s/step - loss: 0.7263 - accuracy: 0.8012 - val_loss: 1.2705 - val_accuracy: 0.6580
Epoch 13/20
350/350 [=====] - 445s 1s/step - loss: 0.7038 - accuracy: 0.8078 - val_loss: 1.2807 - val_accuracy: 0.6563
Epoch 14/20
350/350 [=====] - 449s 1s/step - loss: 0.6833 - accuracy: 0.8145 - val_loss: 1.2916 - val_accuracy: 0.6561
Epoch 15/20
350/350 [=====] - 451s 1s/step - loss: 0.6646 - accuracy: 0.8201 - val_loss: 1.3031 - val_accuracy: 0.6559
Epoch 16/20
350/350 [=====] - 450s 1s/step - loss: 0.6473 - accuracy: 0.8255 - val_loss: 1.3153 - val_accuracy: 0.6551
Epoch 17/20
350/350 [=====] - 451s 1s/step - loss: 0.6314 - accuracy: 0.8304 - val_loss: 1.3279 - val_accuracy: 0.6545
Epoch 18/20
350/350 [=====] - 450s 1s/step - loss: 0.6166 - accuracy: 0.8347 - val_loss: 1.3409 - val_accuracy: 0.6537
Epoch 19/20
350/350 [=====] - 452s 1s/step - loss: 0.6029 - accuracy: 0.8386 - val_loss: 1.3544 - val_accuracy: 0.6534
Epoch 20/20

```

Valores obtenidos con el modelo 3 Epoca 20, nos d^a el valor m^as alto en la epoca 10 Accuracy de : 78.50% y un Valor de Accuracy de 66.03% , comparandolo con la epoca 20 con Accuracy de : 83.86% y un Valor de Accuracy de 65.34%



En las gráficas obtenidas vemos como despues de la epoca 2, empiezan a separarse el entrenamiento del test en nuestra red, en este caso se presenta un mayor overfitting entre ambos valores.

VI. Discusión

Es notable la diferencia entre la comparaci^{on} de modelos hechos y modelos pre-establecidos que nos brinda keras. La desventaja principal que se pod^aba observar era el overfitting en el entrenamiento, puesto que la exactitud se acercaba a casi el 90%, mientras que el valor real de exactitud estaba por debajo del 30%.

Esto debido a la configuraci^{on} de las capas y la gran cantidad de datos que tenemos con CIFAR100, donde era clave tener un mayor n^umero de ^epocas para llegar a este resultado. Sin embargo, al utilizar DenseNet necesitamos una menor cantidad de ^epocas por los par^{am}etros que ya tiene, y nos brinda resultados cercanos de accuracy y val_accuracy, donde vemos que no se disparan los valores y van aprendiendo juntos, y al aplicarle pruebas de predicciones es m^as real el valor a lo que deber^{ia} ser.

Ahora bien, al aumentar las ^epocas obten^{iamos} mejores resultados de entrenamiento, no obstante, tambi^{en} tiene un l^{im}ite y despu^{es} de ahⁱ comienza a decrecer.

La desventaja de la lejanía en los valores de entrenamiento y de exactitud como pasaron en los modelos poco eficientes, es que la red neuronal realmente NO aprende, sino memoriza y esto es contra productivo porque a pesar de que tiene un excelente entrenamiento en un caso de la vida real si moviéramos las imágenes, las rotáramos o cambiáramos su opacidad la red neuronal no podría distinguir ese cambio.

Nuestros valores más altos fueron con el Modelo DenseNet 201, con un 66% de valor en val_accuracy, y accuracy de 78%, presente en la época 10, al aumentar posteriormente las épocas, los valores se van disminuyendo y el entrenamiento va aumentando, haciéndose más notorio el overfitting.

VII. Conclusión

Después de la realización de este producto integrador de aprendizaje, pudimos comprender cómo entrenar una red neuronal artificial para funcionar como un clasificador de imágenes mediante un entrenamiento adecuado. Si bien existen muchos métodos para entrenar una red neuronal artificial la facilidad con la que contamos al usar keras y TensorFlow, nos brinda la posibilidad de enfocarnos solamente a la estructura de la red; saber cuantas capas utilizaremos; en que orden; con que función de activación. Cabe mencionar que Google Collab tiene un límite al usar el GPU que nos brinda, y esto conlleva a estar limitados en la realización de distintas pruebas. Puesto que para cada prueba se necesita un aproximado de 20-30 minutos en promedio.

También existen módulos en las librerías que ayudan a visualizar mejor los datos al testearlos, como lo es tensorboard para la comparación de la gráfica de entrenamiento con el valor de entrenamiento y ver si una red neuronal esta memorizando o si realmente esta aprendiendo y al darle imágenes puede interpretarlo adecuadamente, sin duda alguna conocer este tipo de herramientas nos abre un campo laboral y de investigación que no conocíamos hasta ahora.

Por último, nos gustaría mencionar que gracias a la investigación realizada para la realización del proyecto se cumplió el objetivo propuesto. Al principio tuvimos ciertas dudas y alcanzábamos valores de eficiencia entre 30-40%, después teníamos valores muy separados entre accuracy y val_accuracy; no obstante, gracias a los recursos propuestos en clase se detectó el problema y se le dio solución.

VIII. Referencias bibliográficas

Charte David (2017). Researchgate.Net. Retrieved November 21, 2021, from

https://www.researchgate.net/publication/318888351_Reducción_de_la_dimensionalidad_en_problemas_de_clasificación_con_Deep_Learning_Análisis_y_propuesta_de_herramientas_en_R

AML(2018). Aprendemachinelearning.Com. Retrieved November 21, 2021, from
<https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/>

Mantilla, R(n,d) Redes Neuronales Artificiales, Accelerating the world's research, Departamento de Computación. Universidad de Valparaíso. Retrieved November 21, 2021,

Tsang (2018) Review: DenseNet — Dense Convolutional Network (Image Classification). Retrieved from <https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803>

IX. Link de carpeta de proyecto en su repositorio de Github

<https://github.com/carlosarturotorres/InteligenciaArtificialyRedesNeuronales/tree/main/LasActividadesVanAqui/PIA-Red%20Neuronal%20Convolucional>