**Prácticas Profesionalizantes**

**Tema: Redes Neuronales**

**Bitacora Personal**

**Michel Leandro**

**Sede De Práctica Profesionalizante: UTN - Delta - Campana**

**Responsable a Cargo: Cerrotta Santiago**

**Alumnos: Iván Leiva - Leandro Michel**

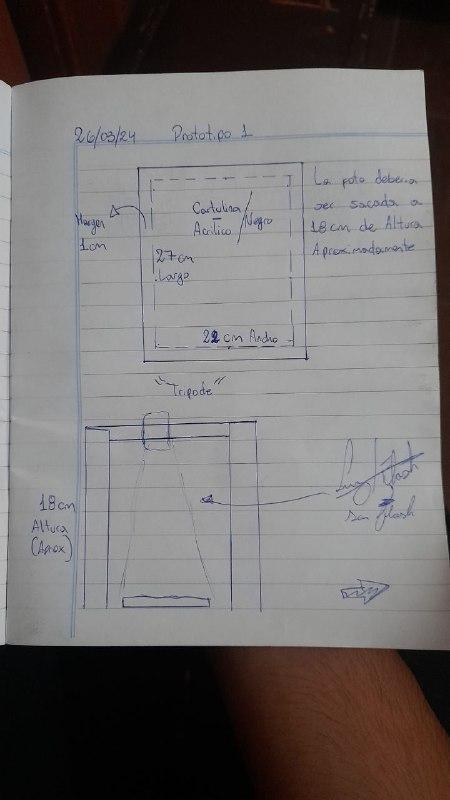
**DESARROLLO**

***26/03/2024***

Iniciamos con la práctica. Conocimos a Santiago y profesores/investigadores involucrados en el estudio de la fotónica.

Dimos un recorrido por el laboratorio y empezamos a introducirnos a los **Objetivos de la práctica –> Investigar sobre redes neuronales mientras aprendemos sobre el tema y concluir con un sistema que dada una imagen, cuente la cantidad de objetos que se encuentran en esta.**

Durante esta práctica pudimos introducirnos a redes neuronales e inteligencia artificial entre otras cosas relacionadas, al igual que empezamos a desarrollar un sistema para sacar fotos desde un mismo ángulo y altura con un celular para que justamente las imágenes sean lo más parecidas posible. Adjunto imagenes:



Además Santiago nos compartió información para que vayamos investigando sobre el tema en los siguientes links.

[**PlayList De Youtube**](https://www.youtube.com/playlist?list=PL-Ogd76BhmcC_E2RjgIIJZd1DQdYHcVf0)

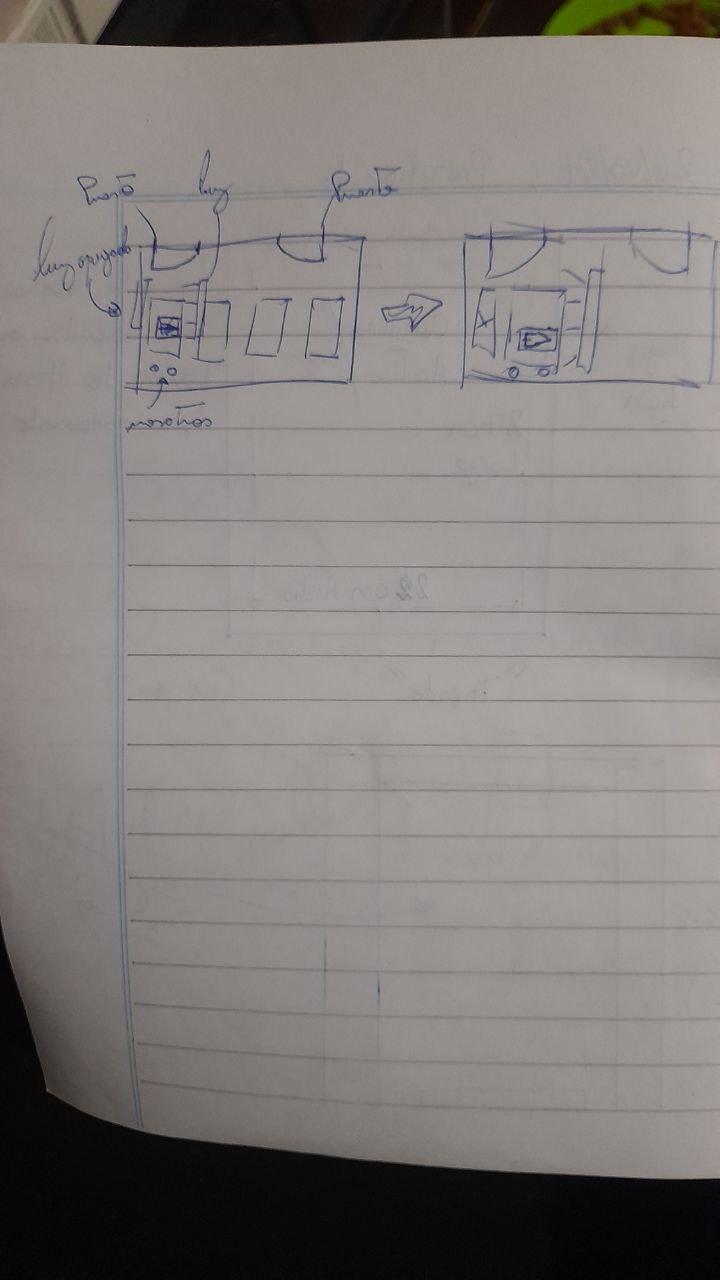
[**Libro De La UTN**](https://ria.utn.edu.ar/handle/20.500.12272/6113)

[**Blog De Una Red Sobre Diabetes**](https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/)

***05/04/2024***

Comenzamos a armar todo el sistema de capturas de fotos. Estas fotos serán con maíces. Para probar, empezamos a sacar fotos a los maíces con un Samsung A21s sobre una cartulina negra.

Empezamos sacando fotos a un único maíz en varias posiciones de la cámara y agregamos un maíz por cada 10 fotos a una n cantidad de maíces hasta llegar a tomar las 10 fotos con 20 maíces.

En total logramos capturar 200 fotos. 



***09/04/2024***

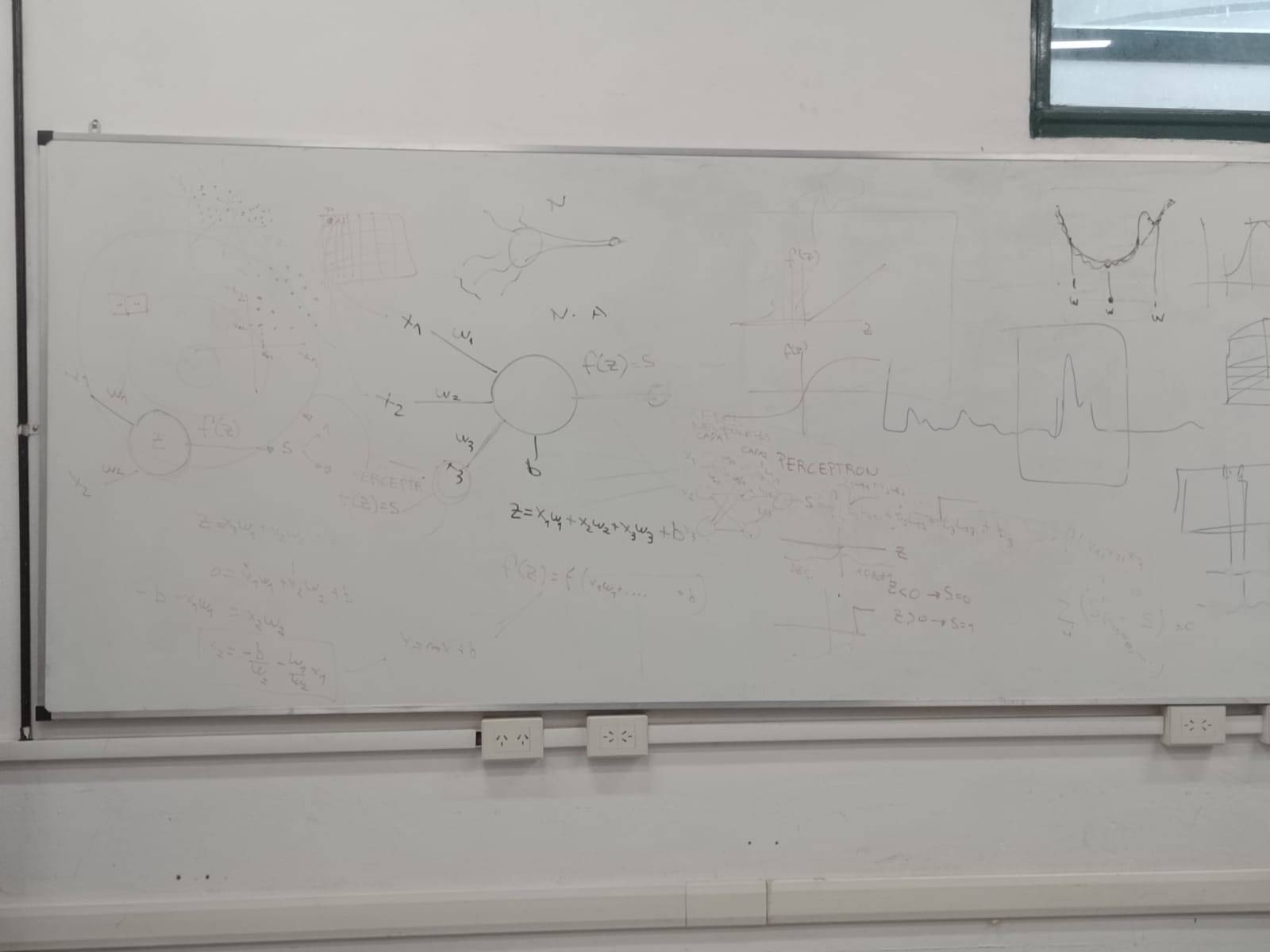
Santiago nos dio una explicacion sobre redes neuronales y inteligencia artificial mientras respondia varias de nuestras preguntas

Continuamos sacando fotos especificamente 40 fotos de 1 a 20 maíces

Además de en un documento google drive compartido clasificar las imágenes en subcarpetas divididas por la cantidad de maíces y también compartir código o archivos de interés mutuo.

Como programas hechos en [**COLAB**](https://colab.research.google.com/), IDLE de Python o Visual Studio para renombrar las fotos segun su “NumeroDeFoto\_CantidadDeMaices”, reescalados en grises y graficador RGB, etc.

**Link:** [Redes Neuronales - Pasantia](https://drive.google.com/drive/folders/1DsEISWzr3MBZR01xsNCRnQ0hTL50S35K)



***12/04/2024***

Volvimos a leer la información y el blog para informarnos un poco más además de experimentar un poco con el código.

También se comenzó a armar el “Trípode” con madera

Acompañamos a Santiago a dar un programa en la radio de la UTN

Luego continuamos sacando fotos 40 fotos de 21 a 29 maíces.



***16/04/2024***

Averiguamos que hace cada linea del codigo del blog, clasificamos y sacamos 10 fotos de 21 a 29 y después 50 fotos de 30

Además hablamos con Santiago sobre arquitecturas, CIFAR-10, subir fotos a python, curva de aprendizaje y cosas a mejorar.

***17/04/2024***

A partir del código del blog: [**Blog De Red Para Detectar Diabetes**](https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/)

Probé diferentes arquitecturas y observé cómo variaba la precisión.

Encontré que la mejor precisión era para la arquitectura:

Además vi que con 2 capas 4-4 neuronas funcionaba muy mal.

***19/04/2024***

Hicimos una búsqueda de [**repositorios**](https://drive.google.com/file/d/15SxlPf80sMQAbMcEopA2lKm53gu2eamQ/view) que podríamos usar pero todos tienen limitaciones que generan más problemas que soluciones y creemos que lo mejor es seguir usando google drive.

Hasta ahora llevamos 50 fotos de 1 a 42 maíces, la idea es estirarnos hasta 50 y después empezar a mezclar maíces con lentejas.

Hice un programa que nos permite reducir la resolución de las imágenes en el caso que lo necesitemos.

***23/04/2024***

Empecé a informarme acerca de clasificación/regresión y CIFAR-10/100 anotando teoría y links en el apartado correspondiente de este documento.

Además de investigar cómo subir nuestras propias fotos a python para entrenar nuestra red neuronal?

[(pt. 1/2) Cómo hacer un clasificador de imagenes desde cero con Tensorflow](https://www.youtube.com/watch?v=EAqb20_4Rdg&ab_channel=AMPTech)

[(pt. 2/2) Cómo hacer un clasificador de imagenes desde cero con Tensorflow](https://www.youtube.com/watch?v=FWz0N4FFL0U&ab_channel=AMPTech)

***24/04/2024***

Segui investigando el blog mencionado anteriormente y separe el código en train and test además de añadir que grafique las curvas de pérdida y precisión mientras me informaba sobre las mismas.

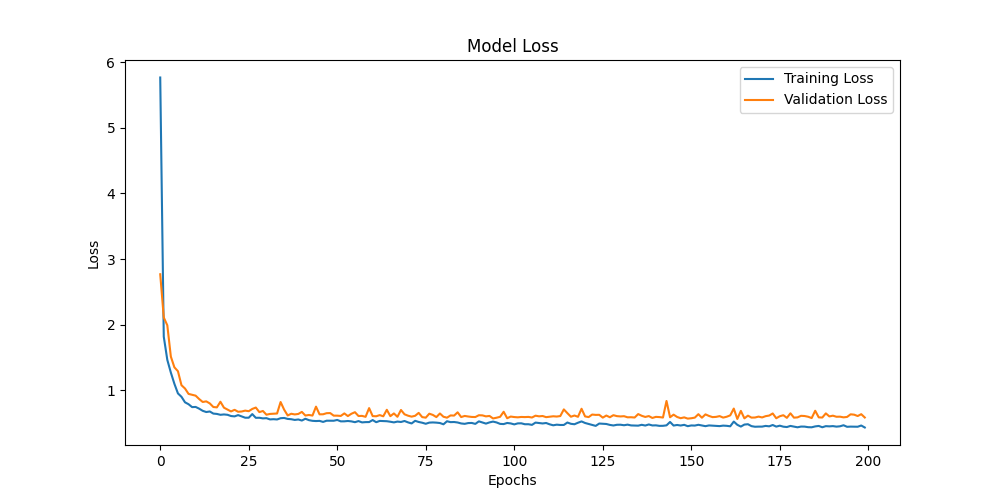
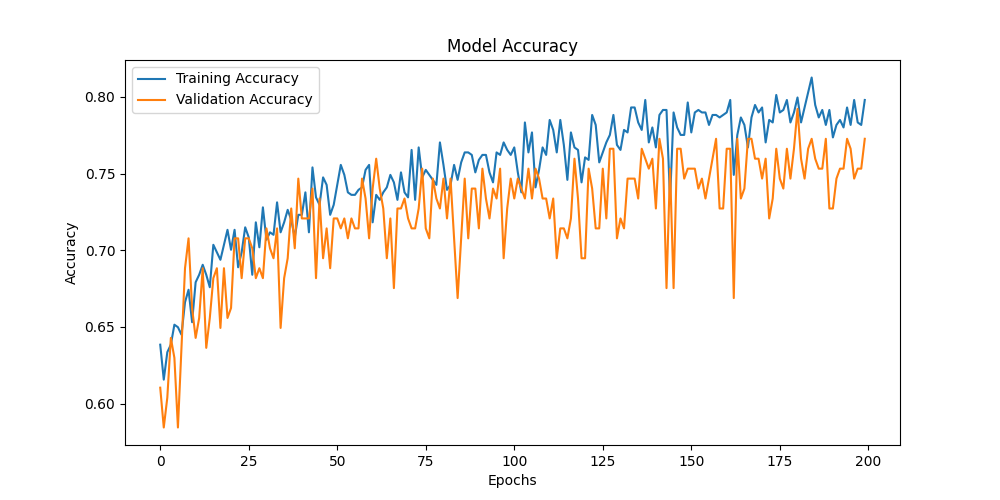
Después comencé a probar diferentes arquitecturas para intentar conseguir un mejor resultado.

Primero intente poner epochs=150 y 4 capas (12-8-4-1) y funcionó mal con un Accuracy: 69.48 a esto se le llama **overfitting** aunque cambiando el 4 por un 6 aumento a 75.97 y eso mismo con epochs=200 da 77.27.

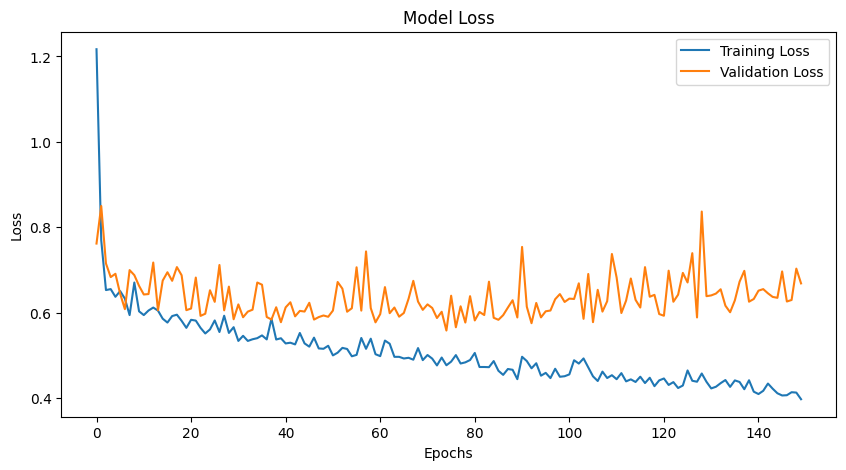
Hasta ahora esta última fue la mejor lograda separada en 80% entrenamiento y 20% testeo.

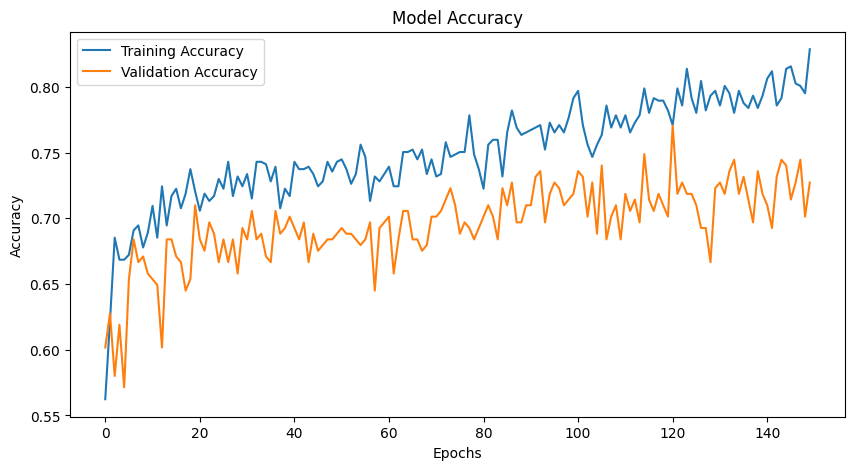
**12 relu -> 8 relu -> 6 relu -> 1 sigmoid: Accuracy: 77.27 (random\_state = 42 - epoch=200)**

**(Random State representa que datos diferentes elige para entrenar)**



**64 relu -> 32 relu -> 16 relu -> 1 sigmoid: Accuracy: 72.73 (random\_state = 67 - epoch=150)**





***26/04/2024***

Terminé de armar el trípode además de pensar formas de optimizarlo como utilizar palitos de crochet para mover los maíces de forma sencilla y colocar pesos encima para más estabilidad.

También concluimos en que sería buena idea atornillar una madera como soporte en la parte de atrás porque de adelante haría sombra en la luz que usamos para tomar las fotos.

Sería buena idea conseguir maderas idénticas a las reglas que usamos para reemplazar las mismas y atornillar las maderas que sostienen esas maderas para más rigidez.

Seguí probando arquitecturas además de haber tenido una charla sobre regresión y clasificación con Santiago.



***29/04/2024***

Investigue formas de guardar y levantar un modelo entrenado de red para no volver a entrenarlo una y otra vez para optimizar el tiempo ya que un entrenamiento de una red un poco compleja podría llevar horas además de entrenar una red con CIFAR y ver la info que mandó Santiago.

Hasta el momento tenemos un total de 2500 fotos de 50 fotos de 1 a 50 maíces.

Mañana comenzamos con fotos de lentejas haciendo 1 maíz con 1 lenteja 10 fotos y así hasta llegar a 1 maíz con 10 lentejas 10 fotos.

En total 100 fotos de 1 a 10 lentejas con 1 maiz.

[Ep. 9 Cómo guardar un modelo entrenado](https://www.youtube.com/watch?v=5X3xWlJ2Ozw&t=145s&ab_channel=AMPTech)

[**Código En GitHub De Red Con CIFAR-10**](https://gist.github.com/eblancoh/d379d92a3680360857581d8937ef114b)

[**Como Entrenar Una Red Con CIFAR-10**](https://datasmarts.net/es/como-entrenar-una-red-neuronal-en-cifar-10-con-keras/)

***06/05/2024***

En resumen lo que estuve haciendo desde el 02/05 hasta hoy sería investigar sobre redes convolucionales lo cual tiene un apartado en este bitácora en la sección de teoría y ademas intente modificar el código de la red de diabetes para que guarde el entrenamiento y poder volver a cargarlo para seguir entrenando calculando el tiempo que tarda en compilar el primer entrenamiento y la carga del entrenamiento previo.

Para esto utilice la librería **time** y **tf.keras.models.save\_model** y **tf.keras.models.load\_model**

El código no presenta errores pero no noto que la precisión mejore, más bien se mantiene en un número fijo sin importar cuantas veces corra el código.

Lo que me hizo pensar en **¿Porque mi precisión no mejora si supuestamente tengo un modelo ya entrenado?** Y busque teoría al respecto la cual también anote en su sección correspondiente.

(Junto con Santiago pude concluir que se debe a que estoy usando los mismos datos o no estoy haciendo cambios a la hora de testear el modelo)

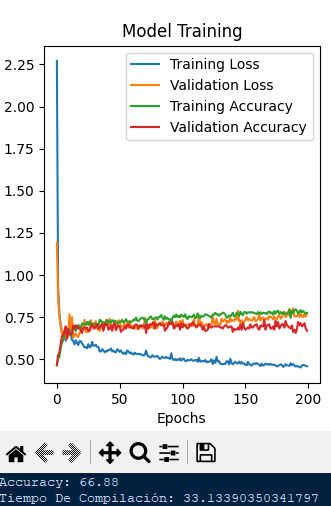
Simplemente continúe intentando algunas veces más hasta que funcionó aumentando prácticamente un 4% de precisión pasando de 66.88 a 72.73.

Por lo cual concluyó que en esta ocasión pudo haber sido un error de python o del sistema.

Mañana vamos a empezar a sacar fotos de 2 maíces hasta 10 lentejas.

El objetivo sería tener 500 fotos de 1 a 10 maíces con 1 a 10 lentejas.

**PRIMER ENTRENAMIENTO**



**ENTRENAMIENTO CON MODELO CARGADO**



***09/05/2024***

Busque las diferencias entre guardar un modelo en formato .keras y .h5 lo cual concluye en ser una cuestión de compatibilidad con el entorno de trabajo, básicamente si vas a utilizar varios tipos de librerías es conveniente usar .h5 ya que es más “universal” por lo contrario si solamente trabajaremos en entorno keras, usar .keras seria mejor opción por la compatibilidad con las mismas librerías.

Comencé a leer el blog que mandó Santiago y anote conceptos sobre diferentes tipos de capas y cuestiones sobre las arquitecturas.

***13/05/2024***

Investigue sobre [**ImageNET y AlexNET**](https://lamaquinaoraculo.com/deep-learning/alexnet/)y anote puntos clave.

Además de leer los documentation de algunas capas de CNN y anotar para que sirven, junto con otro [**Blog De CNN**](https://es.w3d.community/hector/una-guia-comprensiva-para-las-redes-neuronales-convolucionales-la-forma-eli5-hp4) que nos paso Santiago.

Experimente más con el código para entrenar una CNN con CIFAR-10 dejando solamente 2 capas Conv2 y cambiando filtros, kernels, epochs, poolings y batch size, anotando los porcentajes, tiempos y características de cada arquitectura probada resaltando cada cambio.

| **ARQUITECTURA** | **PRECISIÓN/PERDIDA** | **TIEMPO/EPOCHS/BATCH** |
| --- | --- | --- |
| (Conv2D(filters=32)  kernel\_size=(3, 3)  (MaxPooling2D(pool\_size=(2, 2)))  (Dropout(rate=0.25))  (Conv2D(filters=64)  kernel\_size=(3, 3) | Test loss: 0.8973486423492432  Test accuracy: 0.692300021648407 | 10min 17s  batch\_size=128, epochs=5 |
| (Conv2D(filters=32)  kernel\_size=(3, 3)  (MaxPooling2D(pool\_size=(2, 2)))  (Dropout(rate=0.25))  (Conv2D(filters=64)  kernel\_size=(3, 3) | Test loss: 0.7938066124916077  Test accuracy: 0.7232000231742859 | 19min 39s  batch\_size=128, epochs=10 |
| (Conv2D(filters=64)  kernel\_size=(6, 6)  (MaxPooling2D(pool\_size=(3, 3)))  (Dropout(rate=0.25))  (Conv2D(filters=128)  kernel\_size=(6, 6) | Test loss: 0.8057354688644409  Test accuracy: 0.722000002861023 | 51min 41s  batch\_size=128, epochs=10 |
| (Conv2D(filters=64)  kernel\_size=(6, 6)  (MaxPooling2D(pool\_size=(6, 6)))  (Dropout(rate=0.25))  (Conv2D(filters=64)  kernel\_size=(6, 6) | Test loss: 1.088913917541504  Test accuracy: 0.6236000061035156 | 10min 40s  batch\_size=64, epochs=5 |
| (Conv2D(filters=128)  kernel\_size=(9, 9)  (MaxPooling2D(pool\_size=(6, 6)))  (Dropout(rate=0.45))  (Conv2D(filters=128)  kernel\_size=(9, 9) | Test loss: 1.1299715042114258  Test accuracy: 0.6039000153541565 | 47min 10s  batch\_size=96, epochs=5 |
| (Conv2D(filters=64)  kernel\_size=(3, 3)  (MaxPooling2D(pool\_size=(6, 6)))  (Dropout(rate=0.45))  (Conv2D(filters=64)  kernel\_size=(3, 3) | Test loss: 1.208124041557312  Test accuracy: 0.5716999769210815 | 11min 57s  batch\_size=148, epochs=10 |
| (Conv2D(filters=64)  kernel\_size=(3, 3)  (MaxPooling2D(pool\_size=(3, 3)))  (Dropout(rate=0.25))  (Conv2D(filters=64)  kernel\_size=(3, 3) | Test loss: 0.7594046592712402  Test accuracy: 0.7386999726295471 | 16min 50s  batch\_size=128, epochs=10 |
| (Conv2D(filters=64)  kernel\_size=(2, 2)  (MaxPooling2D(pool\_size=(3, 3)))  (Dropout(rate=0.25))  (Conv2D(filters=32)  kernel\_size=(2, 2) | Test loss: 0.9426705241203308  Test accuracy: 0.6705999970436096 | 11min 31s  batch\_size=128, epochs=10 |

***16/05/2024***

Entre ayer y hoy además de averiguar sobre CNN “Famosas”, su función, inputs y competencias.

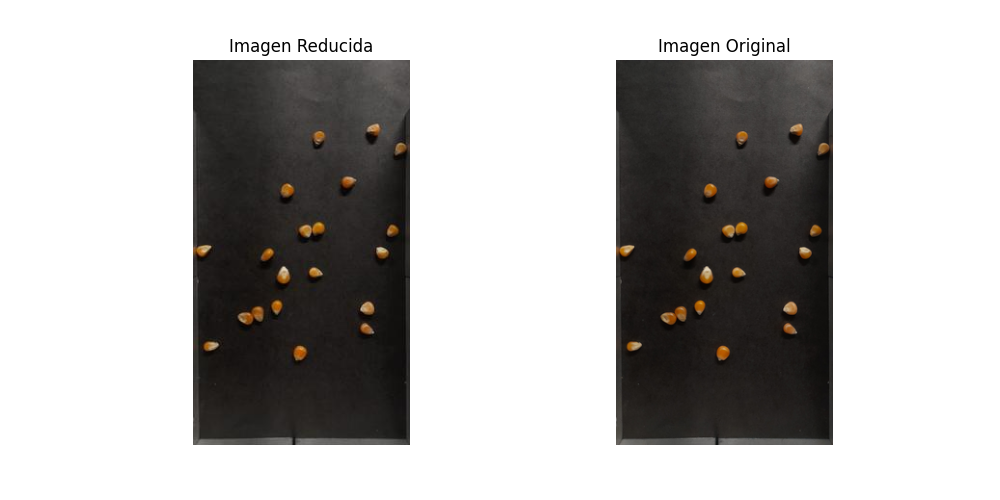
Junto con Ivan empezamos a achicar 10 fotos de 1 a 10 maíces un 40% y a buscar formas de armar nuestro primer dataset inspirado en [**CIFAR-10**](https://www.cs.toronto.edu/%7Ekriz/cifar.html), nos dimos cuenta que a pesar de tener una pequeña base era un poco más complejo de lo que aparentaba y nos ayudamos con [**Blogs**](https://stackoverflow.com/questions/35032675/how-to-create-dataset-similar-to-cifar-10) que nos compartimos para poder llevar a cabo el armado del dataset.

Se concluye que la arquitectura sera Conv2->MaxPooling->Conv2->MaxPooling->Flatten para la primera prueba aunque todavía no logramos correr la red como tal.

***20/05/2024***

Arranque viendo porque la última vez no me permitía subir archivos a colab desde la PC del laboratorio y concluí que fue un error del momento ya que en mi PC me lo permite perfectamente.

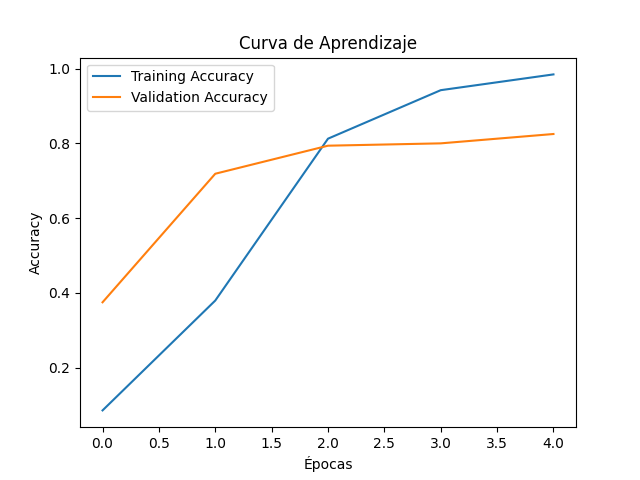
Después arme códigos para visualizar el contenido de un dataset y comparar una imagen pura con una reducida, unifique todo en un mismo código mejor comentado que carga, reduce, compara y arma un dataset.



Separamos train (80%) and test (20%) de un dataset de 800 fotos y entrenamos una red en RGB y grises anotando resultados o diferencias en las siguientes tablas.

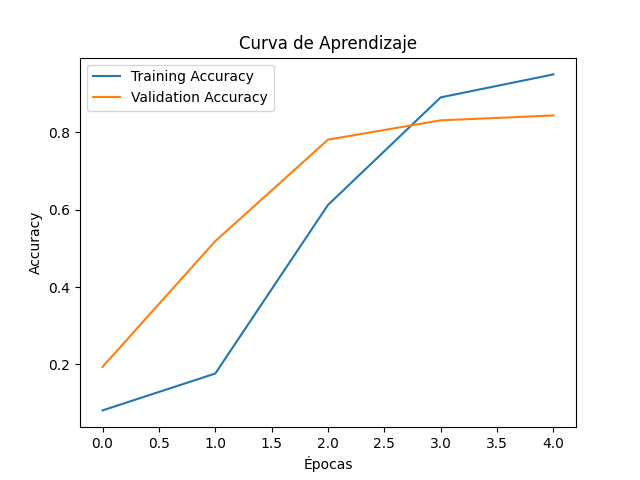
**ESCALA RGB**

| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 12.5mins  Total time: 1h 2 min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 82,50% |
| --- | --- |



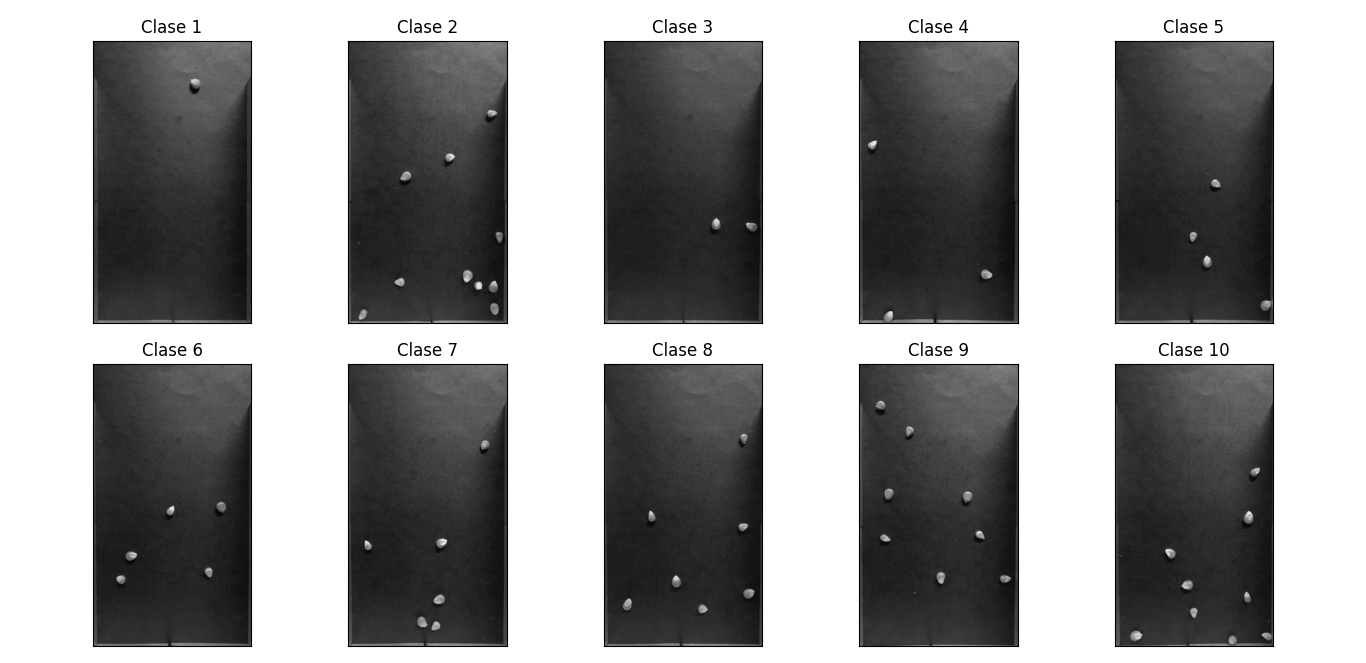
**ESCALA DE GRISES**

| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 10,8mins  Total time: 54mins  num Imagenes: 800 GRISES  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 84,38% |
| --- | --- |



***23/05/2024***

Corregí el código sobre la visualización de los datasets.



Compare una imagen de 50 maíces reducida factor 5(450X800), 10(225X400), 13(173X307) y 15(150X266).

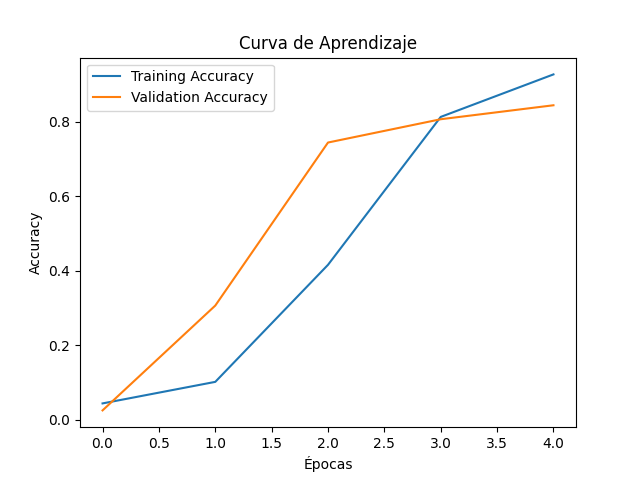


**FACTOR 5(450x800) FACTOR 10(225x400) FACTOR 13(173x307) FACTOR 15(150x266)**

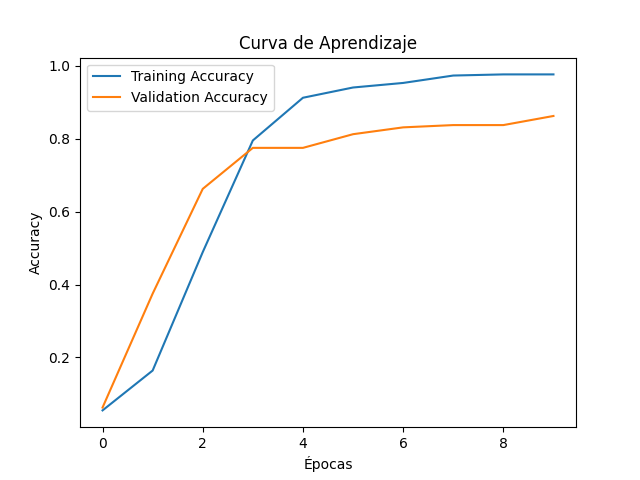
Decidimos entrenar únicamente en RGB y comenzamos a buscar la mejor arquitectura para intentar acercarnos lo mayor posible al 95% de precisión, modificando las capas convolucionales, dropout, max pooling, etc.

**PRUEBAS EN RGB**

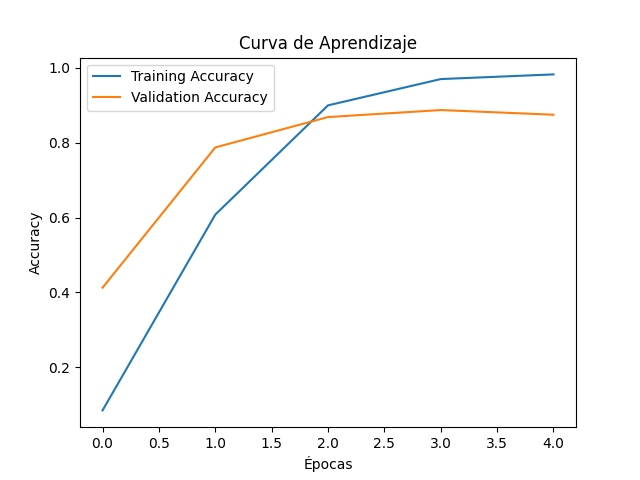
| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dense 256n, relu  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 85s  Total time: 7min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 85,62% |
| --- | --- |



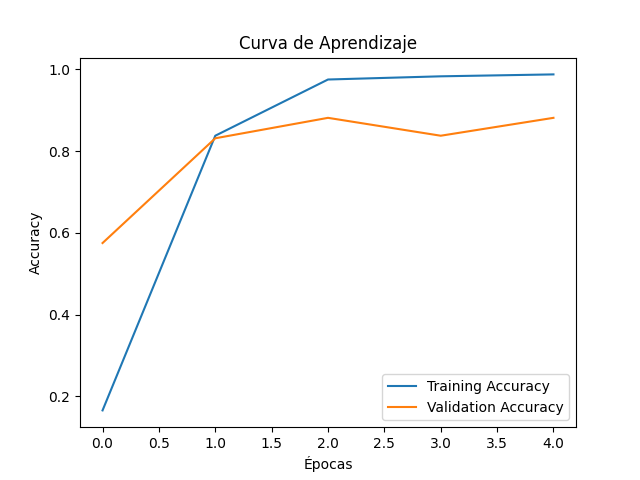
| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 3x3  Dropout 0.25  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 3x3  Dropout 0.25  Flatten  Dense 512n, relu  Dense 256n, relu  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 43s  Total time: 7min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 86,25% |
| --- | --- |



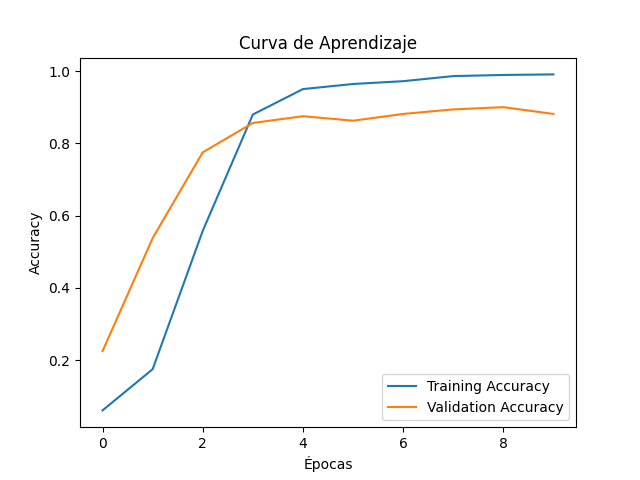
| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  QUITANDO DENSE 256N  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 80s  Total time: 6.66min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 87,50% |
| --- | --- |

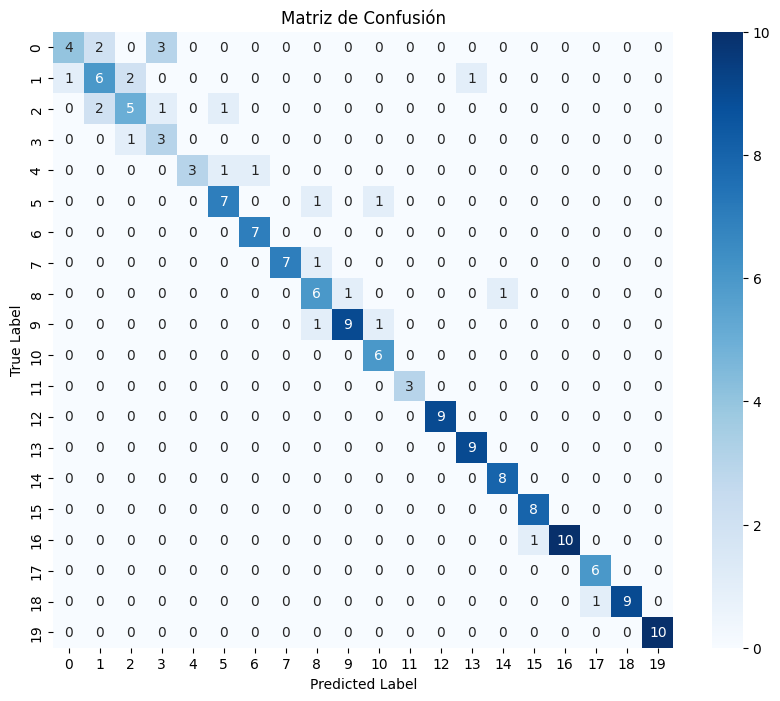


| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  QUITANDO DROPOUTS  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Flatten  Flatten  Dense 512n, relu  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 80s  Total time: 6.66min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 88,13% |
| --- | --- |



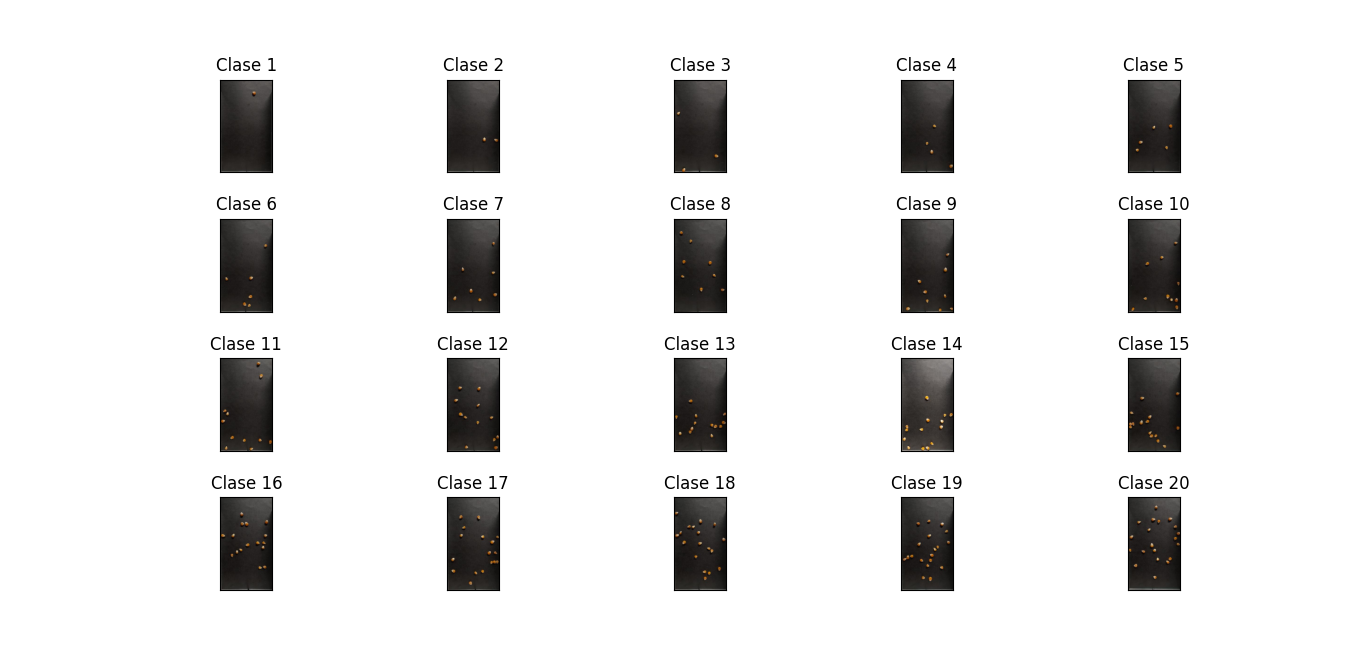
| conv2d, 64f, 3x3, relu  conv2d, 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 128f, 3x3, relu  conv2d 128f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dense 256n, relu  Dropout 0.5  Dense 20n (numClases) softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 5.53min  Total time: 55min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 88,13% |
| --- | --- |





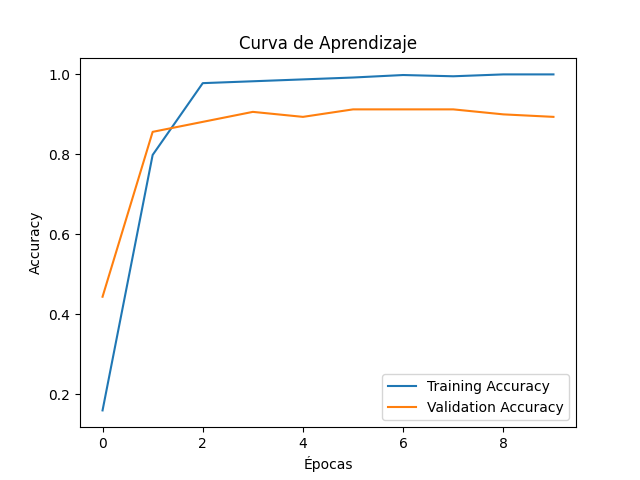
***27/05/2024***

Comencé corrigiendo el código de armado de datasets ya que guardaba mal las clases y no tenian un orden coherente además de hacer cambios en el visualizador.

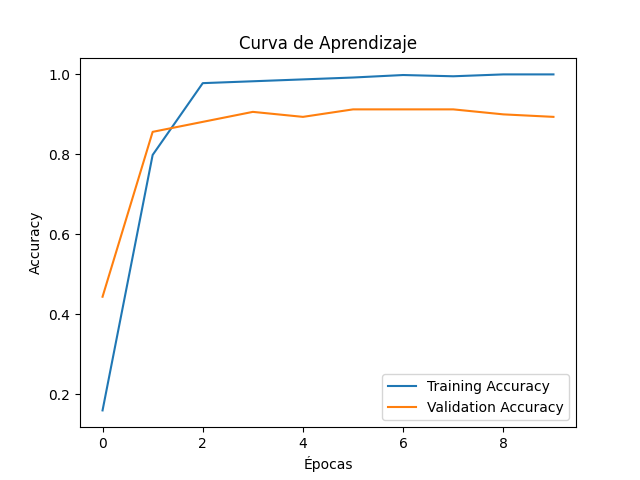


Después me encargue de seguir probando con más y menos capas convolucionales para seguir buscando la mejor arquitectura.

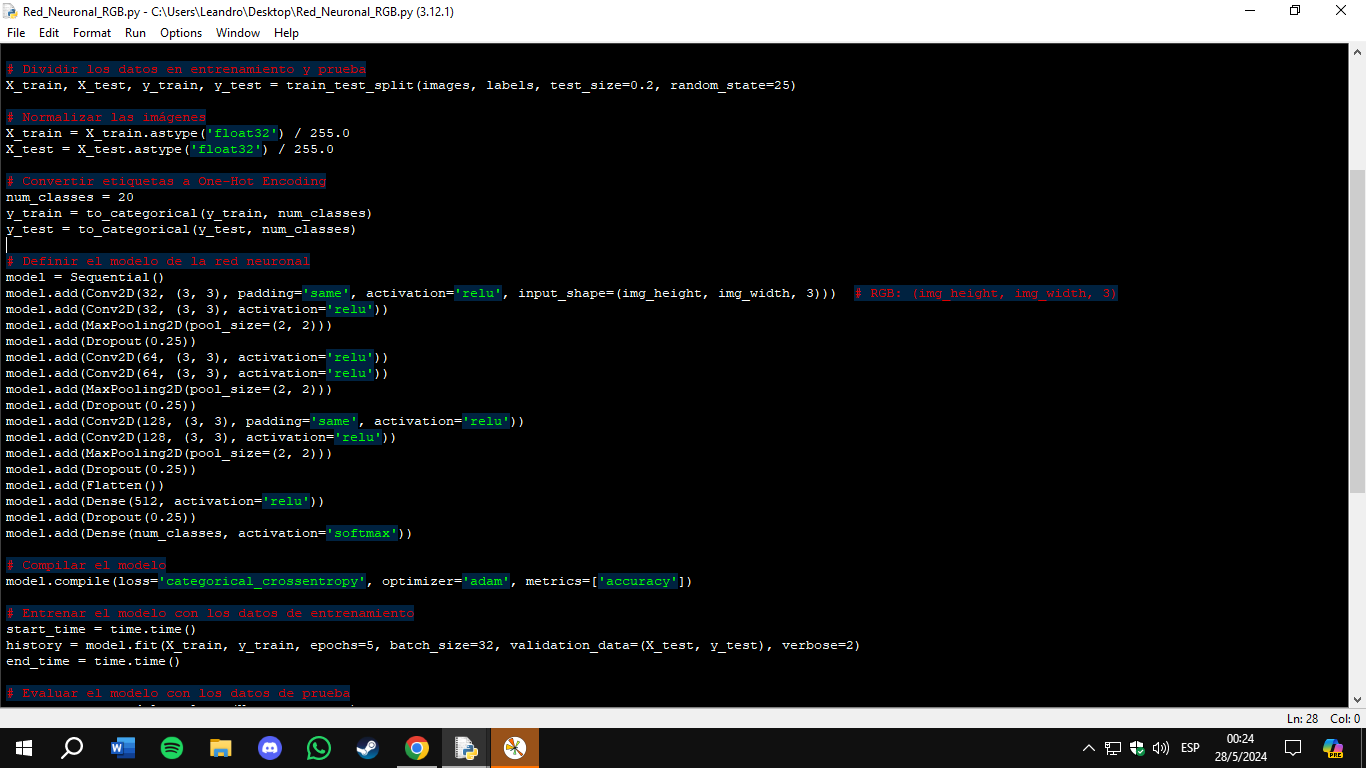
| conv2d, 32f, 3x3, relu  SE QUITARON 2 CONV2D  maxPooling 2x2  QUITANDO DROPOUTS  conv2d 64f, 3x3, relu  maxPooling 2x2  Flatten  Flatten  Dense 512n, relu  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 40s  Total time: 3.33min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 89,38%  Quitando un Flatten y poniendo Dropouts: 87.50% |
| --- | --- |

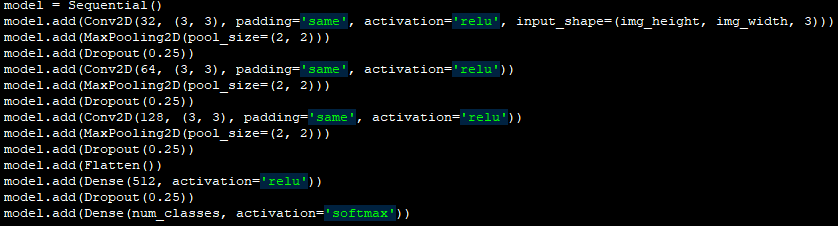


| conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  conv2d, 32f, 3x3, relu  maxPooling 2x2  QUITANDO DROPOUTS  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  conv2d 64f, 3x3, relu  maxPooling 2x2  Flatten  Flatten  Dense 512n, relu  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 40s  Total time: 3.33min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy:  80,62% Con 3 Conv2d  69% Con 4 Conv2d |
| --- | --- |



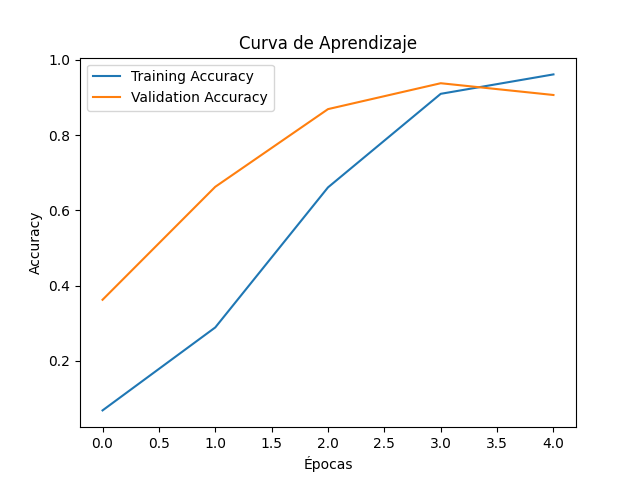
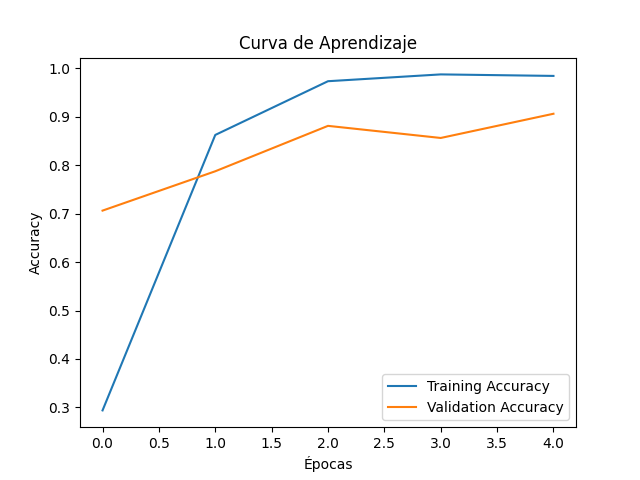
**Poniendo el siguiente tipo de arquitecturas o mayores lo único que logré fue disminuir la precisión por debajo del 80%, por lo que mi hipótesis es que el camino que debemos seguir sería usar no más de 4 capas convolucionales.**





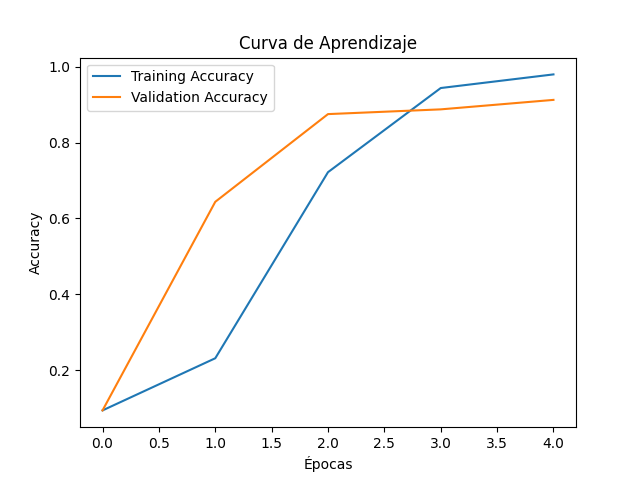
**Un ejemplo de algo conveniente sería usar 2 Conv2d de 32 filtros y 2 Conv2d de 64 filtros separado por su respectivo MaxPooling2D(2, 2) y Dropout(0,25 a 0,5) e ir variando el número de filtros o sacando una capa Conv2d pero no usar más de lo dicho.**

| conv2d, 32f, 3x3, relu  maxPooling 2x2  PODRÍA USARSE DROPOUT 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  PODRÍA USARSE DROPOUT 0.25  Flatten  Dense 512n, relu  PODRÍA USARSE DROPOUT 0.25  Dense 20n (numClases) softmax  (Usando dropouts(0.25) da el mismo resultado)  (Usando 64 y 128 filtros disminuye a %88,75 y tarda casi el triple) | Epochs: 5  BatchSize: 32  Time x Epoch: 39s  Total time: 3.16min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 90,62% |
| --- | --- |



SIN DROPOUT CON DROPOUT

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dropout 0.15  Dense 20n (numClases) softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 41s  Total time: 3.41min  num Imagenes: 800 RGB  resolución: 225alto x 400ancho  num Clases: 20 (40 fts x 20 class)  testAccuracy: 91,25% |
| --- | --- |



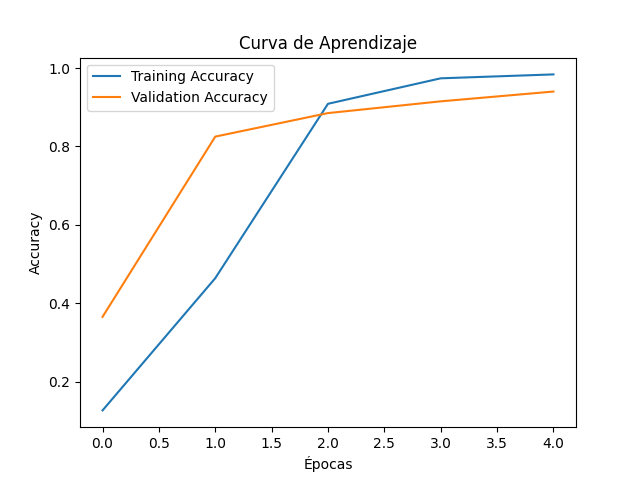
**PROMEDIO DE 5 PRECISIONES: 89,38 + 88,75 + 91,25 + 90,62 + 91,25 = 451,25/5 = 90,25**

Agregue al dataset 200 fotos que serían 10 fotos más por número de maíces entonces quedaría 50 fotos de 1 a 20 maíces con un total de 1000 fotos.

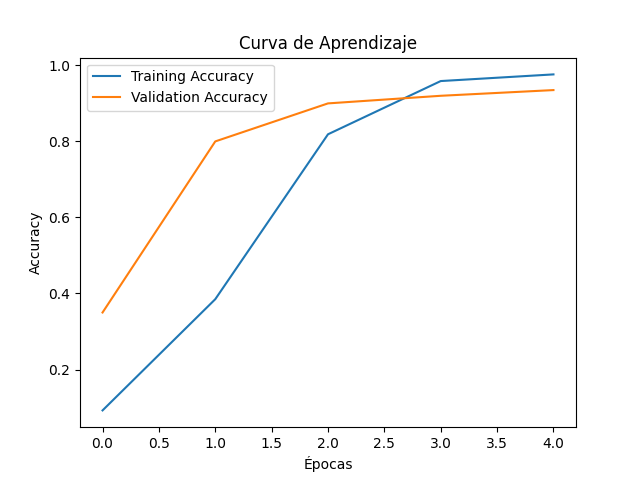
***28/05/2024***

Añadi summary y matriz de confusión del modelo al código de entrenamiento de la red y comencé a correr la mejor arquitectura hasta el momento con el nuevo dataset de 1000 fotos y haciendo cambios a partir del summary.

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  (Usando doble Dense 512n baja a 92,50%)  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 53s  Total time: 4.41min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 94,00% |
| --- | --- |



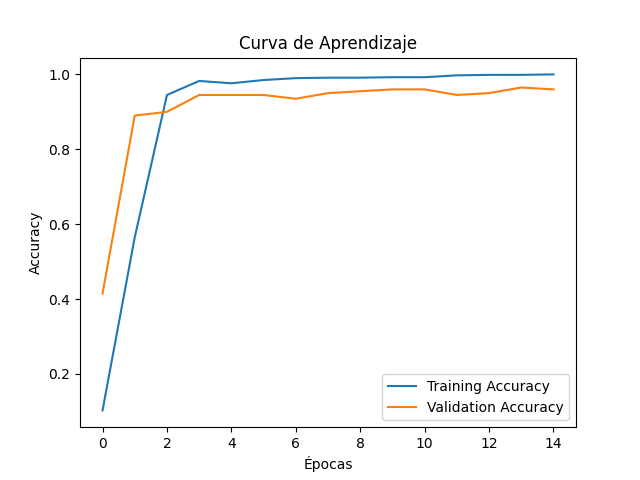
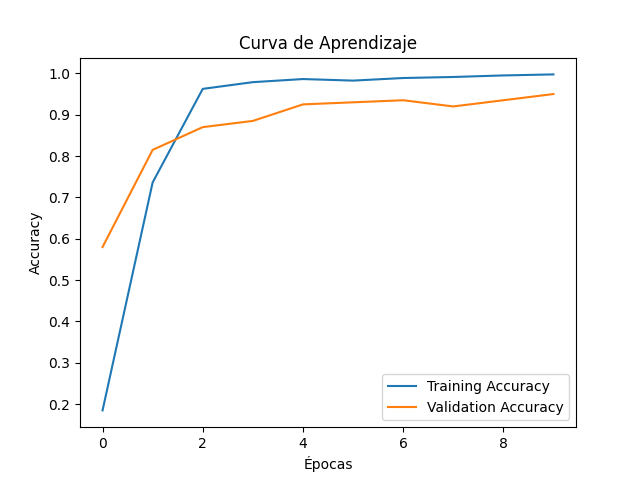
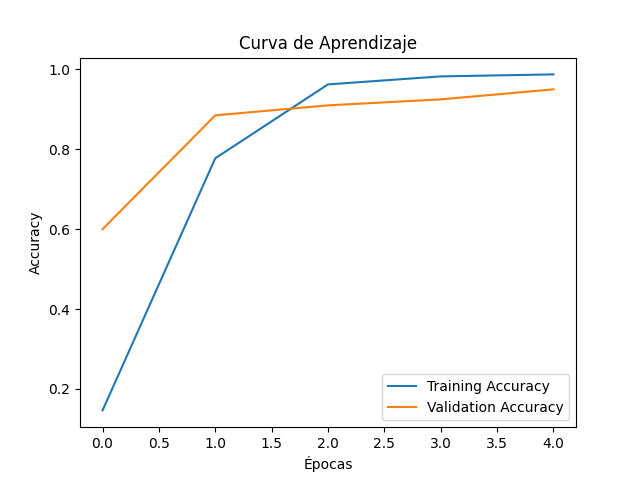
| conv2d, 16f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 512n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 22s  Total time: 1.83min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 93,50% |
| --- | --- |



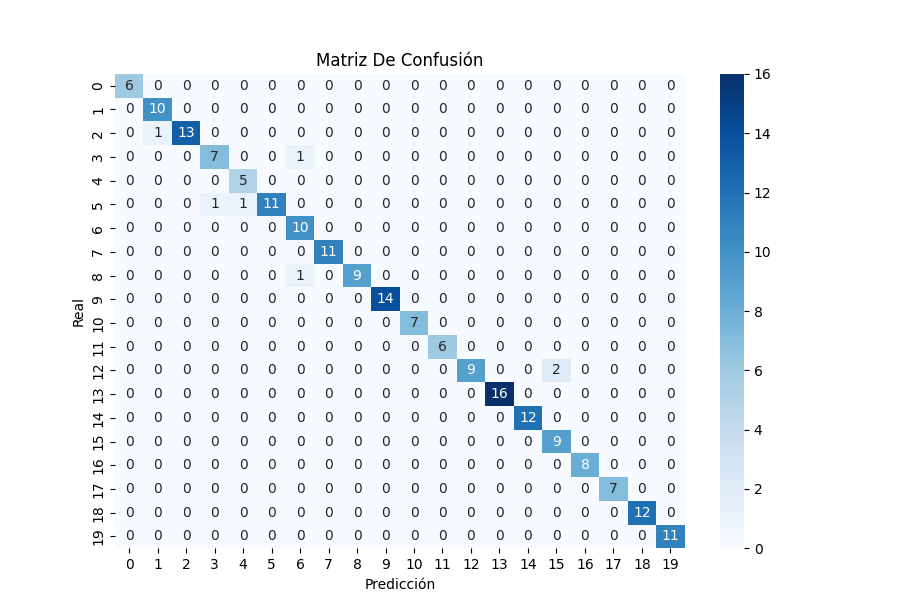
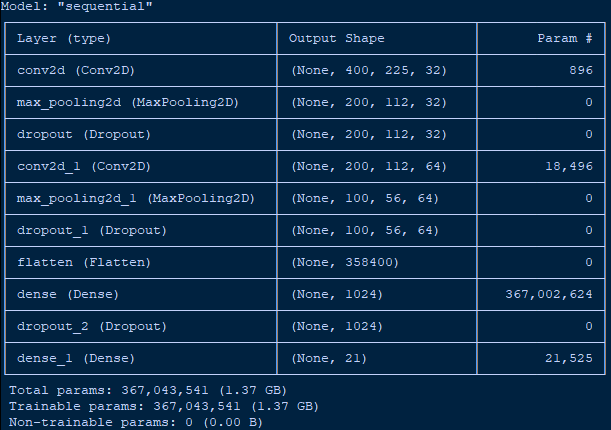
| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dense 512n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 83s  Total time: 6.9min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 91,50% |
| --- | --- |



| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  (Usando Dense 2048n Crasheo)  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 78s  Total time: 6.5min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 95,00% |
| --- | --- |



**5 EPOCH: 95,00%** **10 EPOCH: 95,00%** **15 EPOCH: 96,00%**

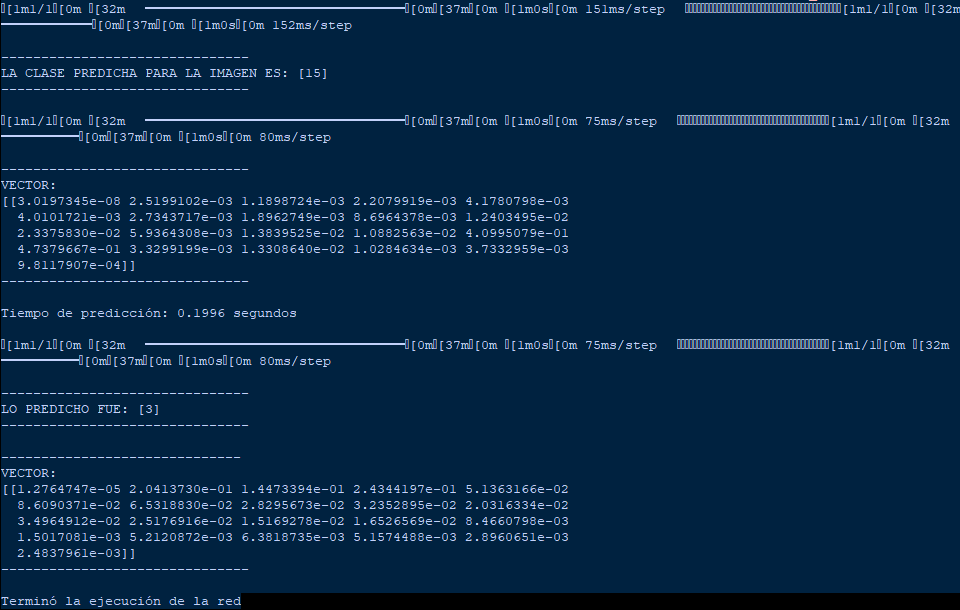


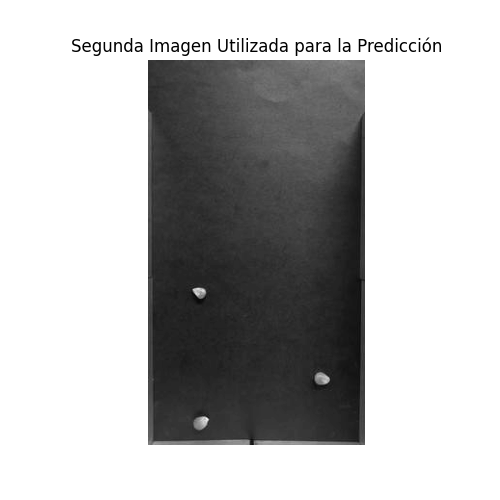
**Tengo la hipótesis de que esta podría llegar a ser la mejor arquitectura ya que en las curvas se observa como la validación deja de tener una tendencia ascendente y sigue de una forma más recta, lo que nos hizo pensar que a partir de este punto la única forma de mejorar la precisión sería usando un conjunto de datos con mas imagenes.**

Corregimos el código de predicciones agregando el vector [1\_CantidadMaices] y graficando las imágenes usadas para predecir.

Se lee de la siguiente forma, el vector es [20] ya que tenemos hasta 20 maíces y cada número es la probabilidad de que en esa imagen subida haya el número de maíces al que hace referencia la posición del vector (Posición 1=1 Maíz - 2=2 Maíces - 20=20 Maíces).

**El número se lee de esta forma, ejemplo tenemos 4.0995079e-01, e-01 representa la cantidad de ceros que hay antes del primer número (4) por lo cual el número del vector en realidad sería 0.40995079 y así con los demás casos.**



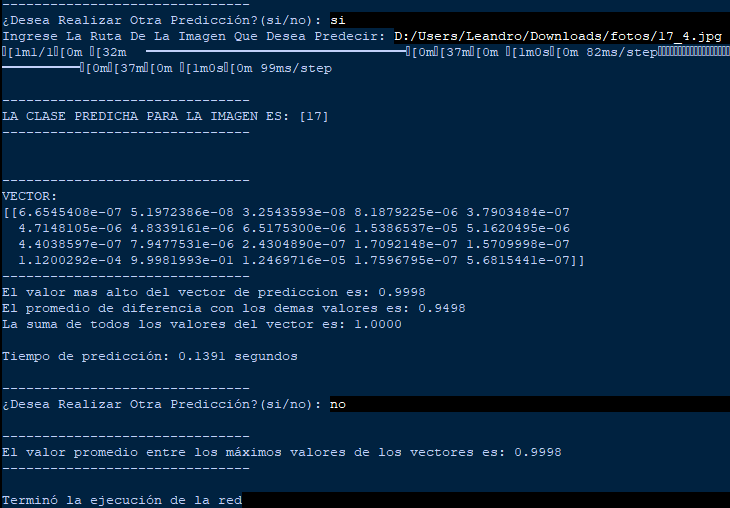
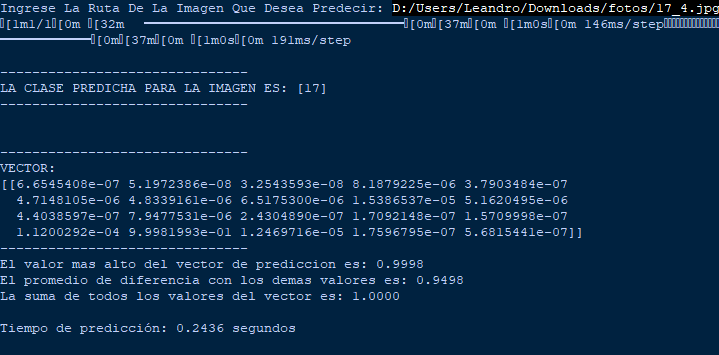


***04/06/2024***

Corregí el código de predicciones ya que en el vector habían 21 valores y debía haber 20.

Ahora al principio del código pregunta la ruta de la imagen a predecir y una vez hecha la predicción se pregunta si se desea hacer otra y así sucesivamente hasta que se ingrese ‘no’.

Además de calcular el mayor número de cada vector, la diferencia en promedio con el resto de números, la sumatoria de todos los números y al final el promedio de todos los números mayores de cada vector.





Predecí 100 imágenes (5 imágenes por cada número de maíces) usando un modelo con 93% de precisión y me dio el siguiente promedio de entre todos los valores máximos.



Dio un resultado equivocado en los siguientes casos.

3 Maíces = Resultado 2 — 6 Maíces = Resultado 5 — 11 Maíces = Resultado 10

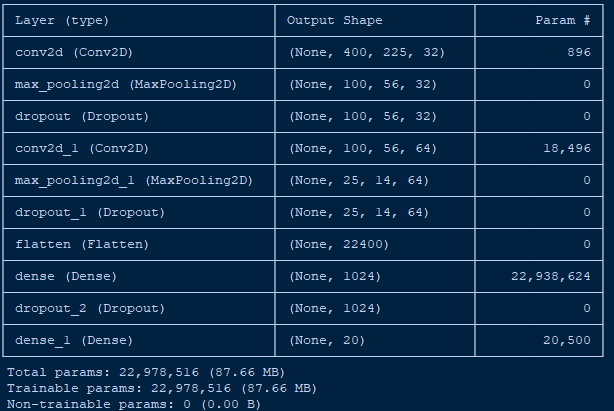
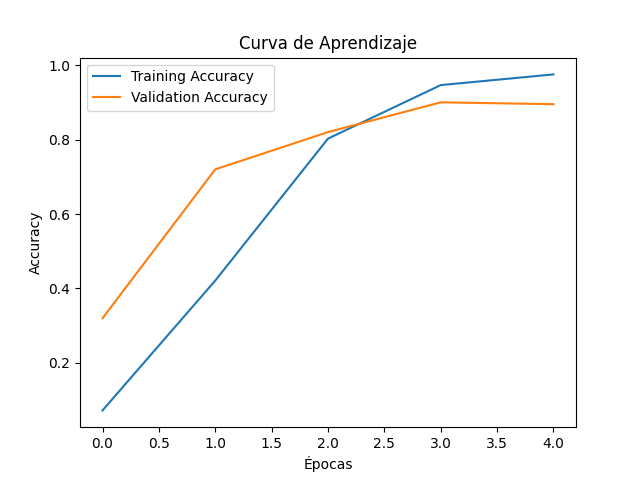
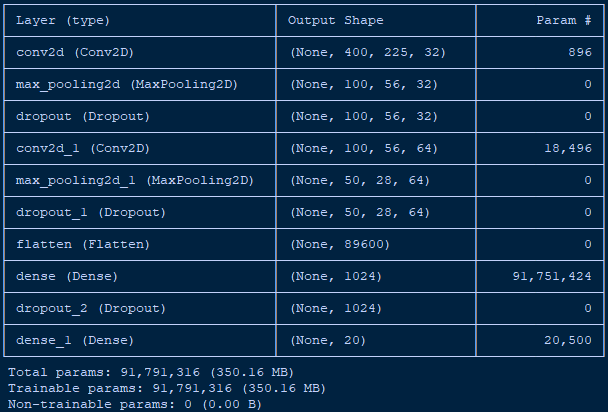
15 Maíces = Resultado 14(2 veces) — 16 Maíces = Resultado 15 — 17 Maíces = Resultado 16

Estos resultados los dio en la primera imagen de un número de maíces a otro.

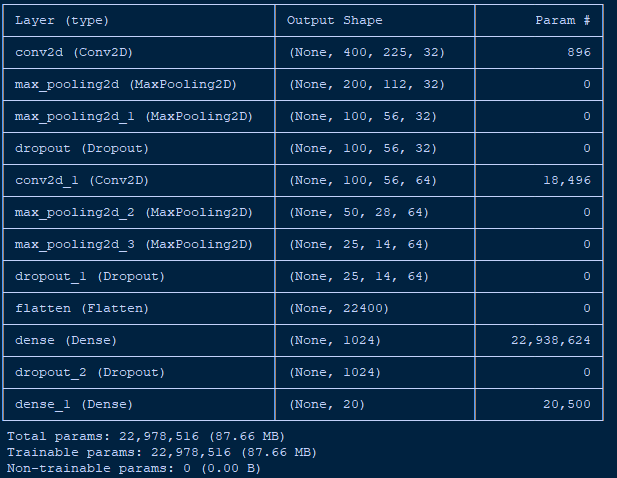
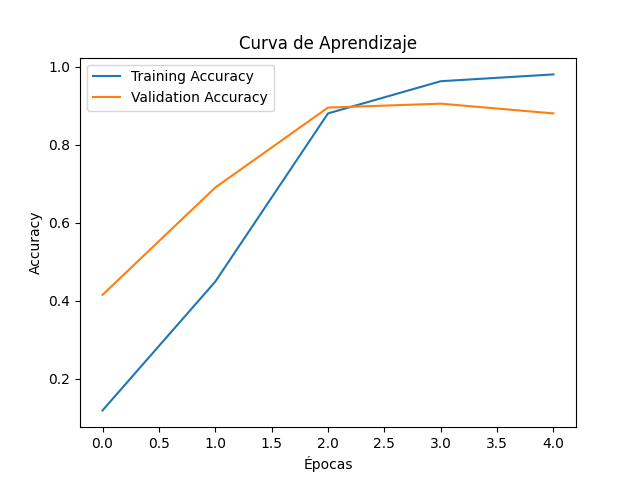
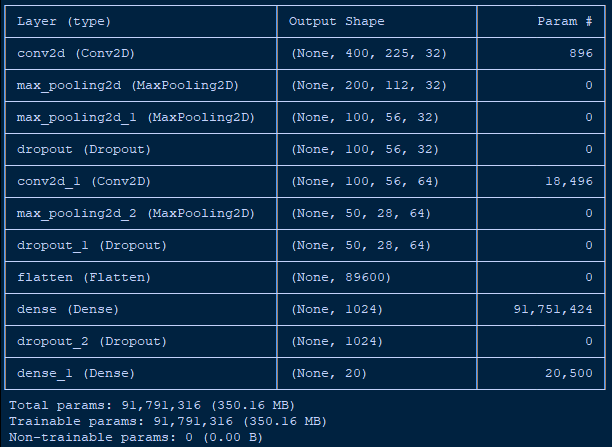
Ejemplo al pasar de una imagen de 2 a 3 maíces dio como resultado 2 y así con los demás casos.

A partir de la mejor arquitectura comencé a editar los maxpooling en busca de mejores resultados.

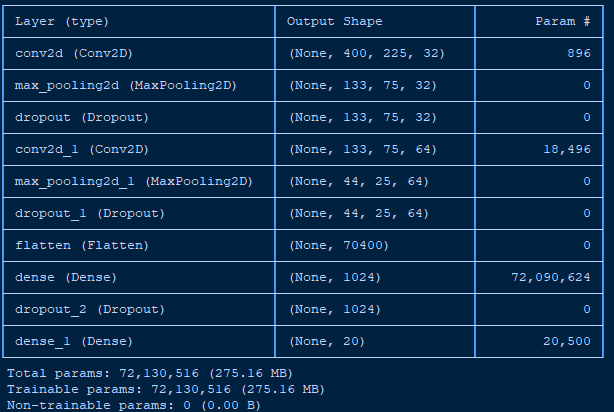
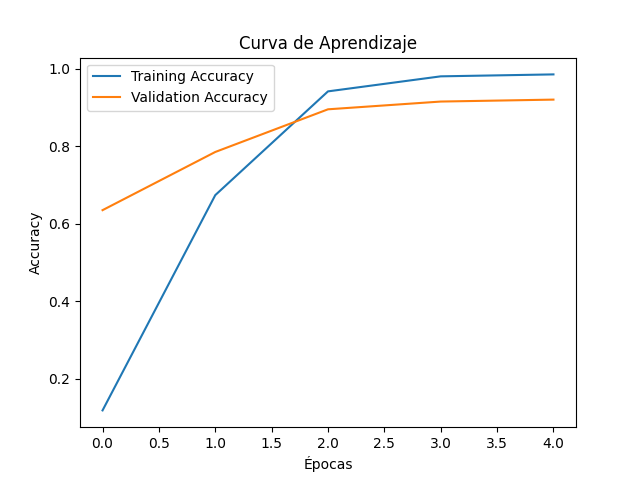
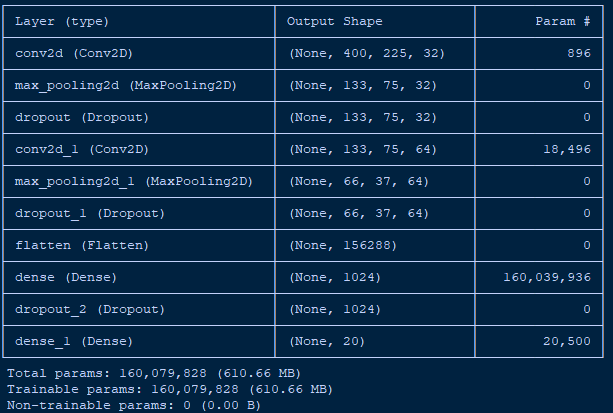
| conv2d, 32f, 3x3, relu  maxPooling 4x4  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 25s  Total time: 1.93min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 89,50%  Flatten: 89600  Ambos maxPooling 4x4: 86%  Flatten: 22400 |
| --- | --- |



| conv2d, 32f, 3x3, relu  maxPooling 2x2  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax  **(No se conocen arquitecturas que usen doble maxPooling seguido)** | Epochs: 5  BatchSize: 32  Time x Epoch: 25s  Total time: 1.93min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 88%  Flatten: 89600  Doble maxPooling 2x2 en ambos: 84.50%  Flatten: 22400 |
| --- | --- |



| conv2d, 32f, 3x3, relu  maxPooling 2x2, strides 3,3  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 25s  Total time: 1.93min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 92%  Flatten: 156288  Ambos strides 3,3: 91.50%  Flatten: 70400 |
| --- | --- |



No encontré mejoras y probando otros casos la precisión bajaba todavía más.

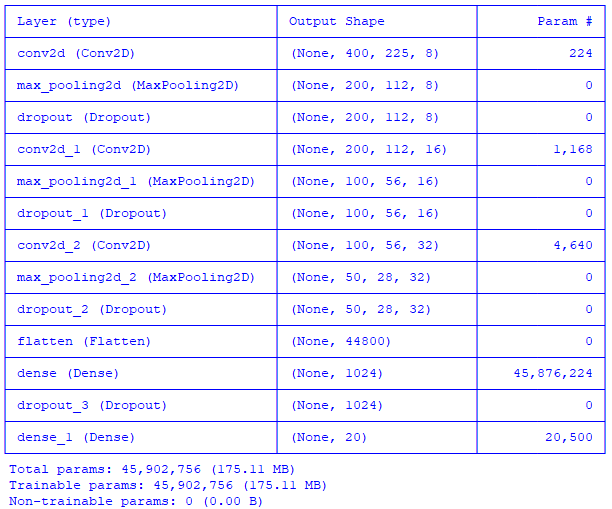
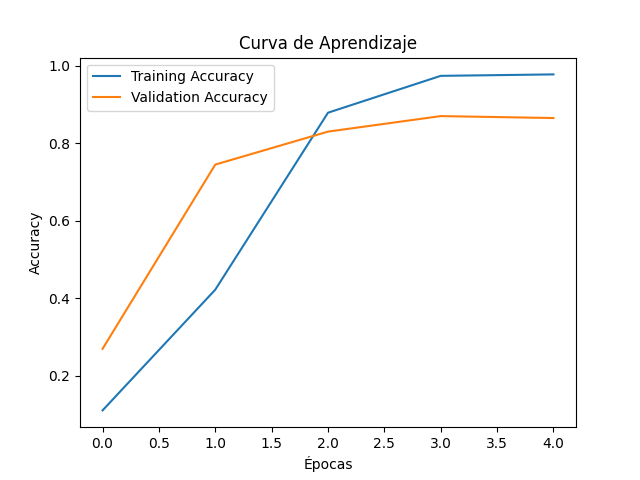
***04/06/2024***

Hasta el momento contamos con 1000 fotos de 1 a 20 maíces (50 fotos por número de maíz) y 2000 fotos más de 1 a 20 maíces de 1 a 10 lentejas (100 fotos por número de maíz de 1 a 10 lentejas, serían 10 fotos por número de lenteja) a partir de esto decidimos empezar con arroz.

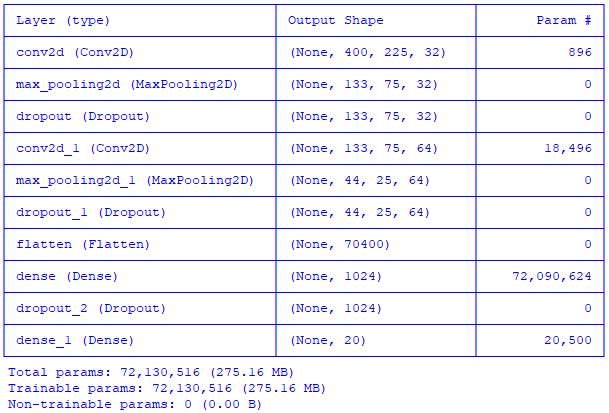
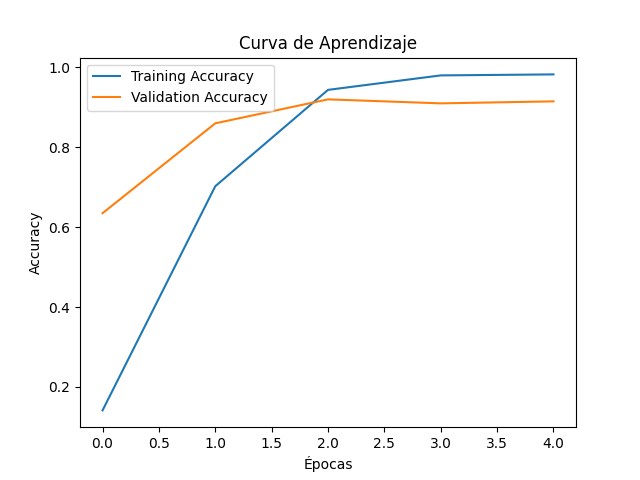
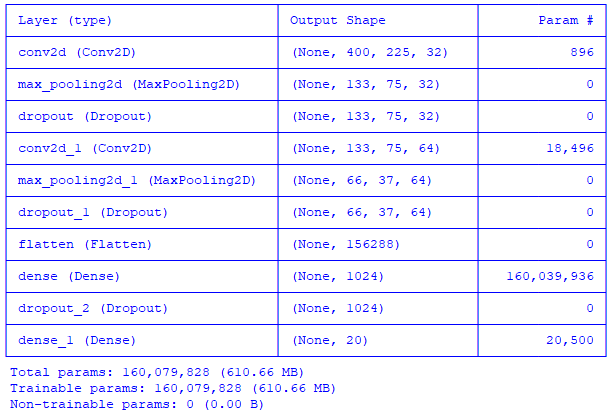
Con Santiago nos dimos cuenta que cuando no se define el stride, usa las medidas del pool size por lo que significa que usar pool size 2x2 nos haría tener stride 2,2 y así en demás casos.

Por lo cual hice nuevas pruebas con maxpooling y con algunos casos que Santiago me pidió.

| conv2d, 8f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d, 16f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 15s  Total time: 1.15min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 86,50%  Flatten: 44800 |
| --- | --- |



| conv2d, 32f, 3x3, relu  maxPooling 3x3  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 46s  Total time: 3.80min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 92%  Flatten: 156288  10 Epoch: 93,50%  Ambos maxPooling 3x3: 90%  Flatten: 70400 |
| --- | --- |



Hasta ahora la mejor arquitectura que encontré fue la mencionada los días pasados a la cual aplicando data augmentation haciendo volteos horizontales y verticales quedan 3000 fotos pero esto en vez de dar mejores resultados hacer que la precisión baje o simplemente se mantenga cerca del porcentaje actual que tenemos.

***06/06/2024***

La mayoría de cambios más significativos en la arquitectura ya fueron testeados así que tengo la teoría de que en este punto solamente me quedaría hacer cambios en batch size, random state, epochs o tamaño del kernel de las capas convolucionales en la mejor arquitectura.

Por defecto: kernel 3,3 - random state 42 - batch size 32 - epoch 5

Ambos Kernel: 3, 3 - random state: 42 - batch size: 32 - 91,50%

Ambos Kernel: 6, 6 - random state: 42 - batch size: 32 - 91,50%

Ambos Kernel: 9, 9 - random state: 42 - batch size: 32 - 91,50%

Ambos Kernel: 3, 3 - random state: 62 - batch size: 32 - 93%

Ambos Kernel: 3, 3 - random state: 82 - batch size: 32 - 87.50%

Ambos Kernel: 3, 3 - random state: 22 - batch size: 32 - 92%

Ambos Kernel: 3, 3 - random state: 12 - batch size: 32 - 91%

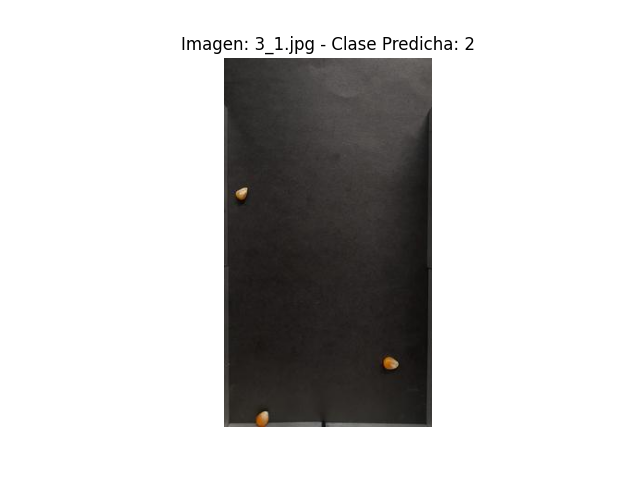
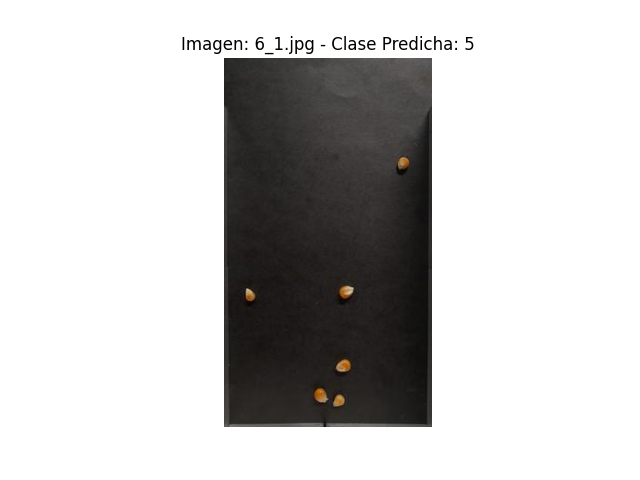
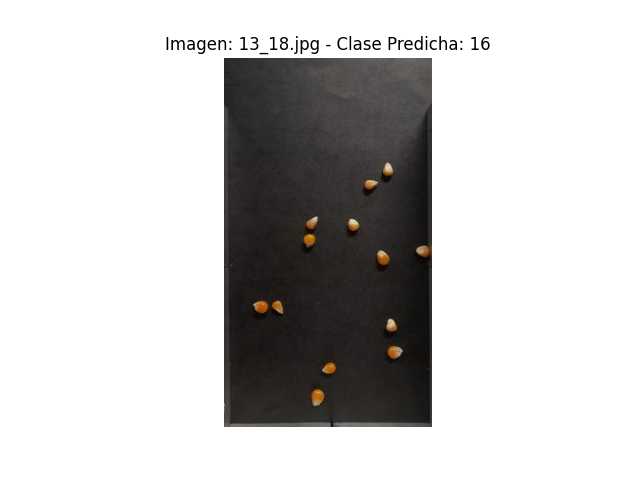
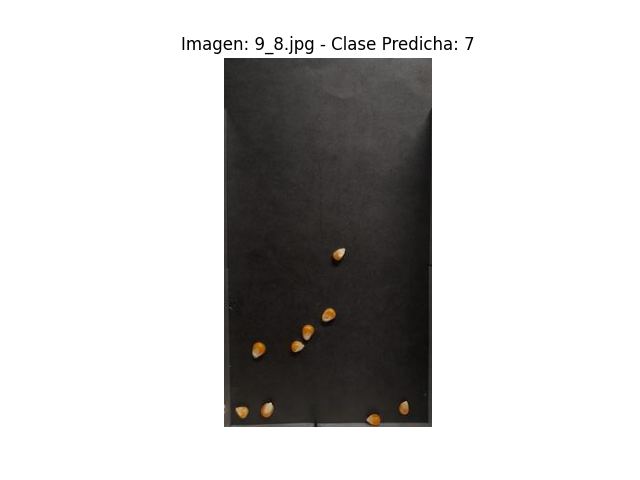
Ambos Kernel: 3, 3 - random state: 42 - batch size: 52 - 90,50% (Poniendo 10 epoch da 95%)

Ambos Kernel: 3, 3 - random state: 42 - batch size: 12 - 93%

(Poniendo 15 epoch da 96.50% tardó 41min y con más comenzó a oscilar entre ese valor)

***10/06/2024***

identifique las predicciones incorrectas de la red con 96.5% de precisión de solo maíces utilizando el dataset entero con 1000 fotos de maices solos.



Precisión de predicción promedio con dataset de maíces solos con modelo de solo maíces.



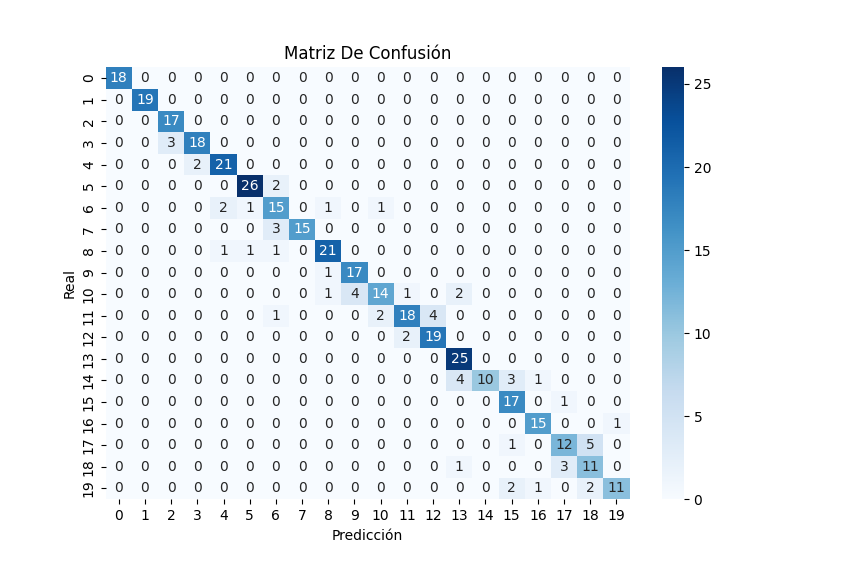
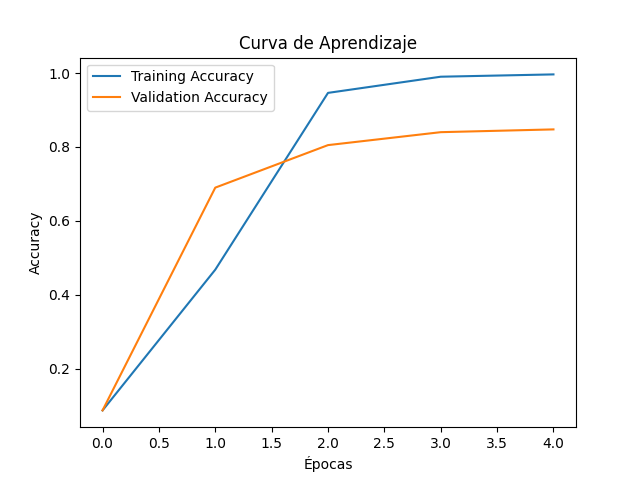
Precisión de predicción con dataset de maíces y lentejas del modelo con 96.5% de solo maíces.

Una observación es que con poca cantidad de maíces dio buenas predicciones pero al ir aumentando la cantidad de matices empezó a decaer.



Entrenamiento de la mejor arquitectura con dataset de maíces con lentejas.

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 160s  Total time: 13,33min  num Imagenes: 2000 RGB  resolución: 225alto x 400ancho  Maíces Con Lentejas  num Clases: 20 (100 fts x 20 class)  testAccuracy: 84,50% |
| --- | --- |

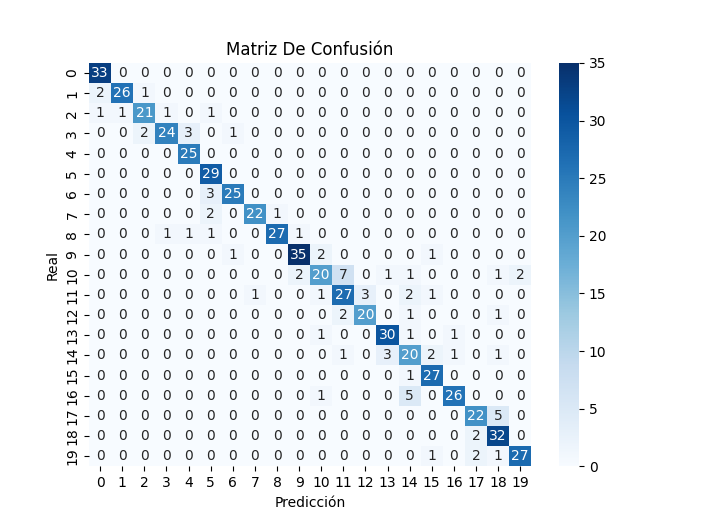
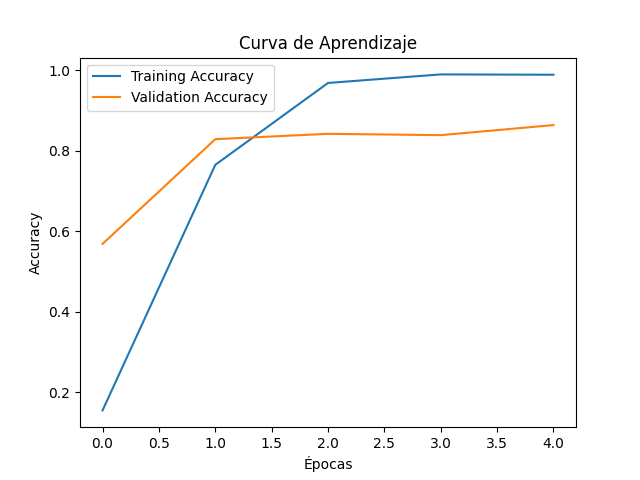


Precisión de predicción con dataset de maíces con lentejas de modelo con maíces y lentejas.



Entrenamiento de la mejor arquitectura con dataset de maíces con lentejas y maíces solos.

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 245s  Total time: 20.41min  num Imagenes: 3000 RGB  resolución: 225alto x 400ancho  Maíces Solos y Con Lentejas  num Clases: 20 (150 fts x 20 class)  testAccuracy: 86,33%  Quitando Dropouts: 89,50% |
| --- | --- |



Precisión de predicción con dataset de maíces con lentejas y maíces solos de modelo con maíces con lentejas y maíces solos.



***17/06/2024***

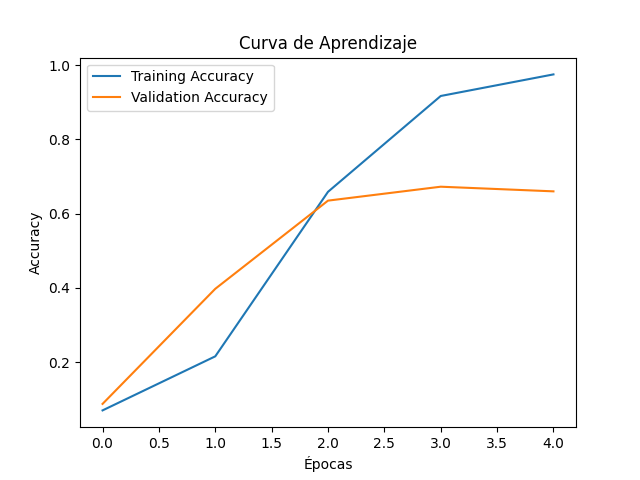
Calcule la precisión de predicción del dataset de maíces y lentejas con el modelo con 96.5% de precisión con solo maíces.



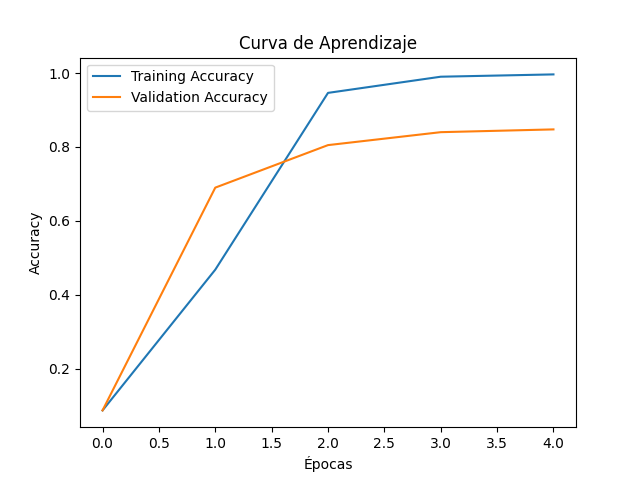
**La red confunde muchísimo las lentejas con los maíces. La red entrenada anda muy bien para el caso particular anterior.**

Hice cambios en capas y filtros entre otras cosas de la arquitectura para intentar mejorar la precisión entrenando con maíces y lentejas.

| conv2d, 16f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 46s  Total time: 3.83min  num Imagenes: 2000 RGB  resolución: 225alto x 400ancho  Maíces Con Lentejas  num Clases: 20 (100 fts x 20 class)  testAccuracy: 66%  Conv32-64-128: 74,25%  Menos filtros bajaba la precision |
| --- | --- |

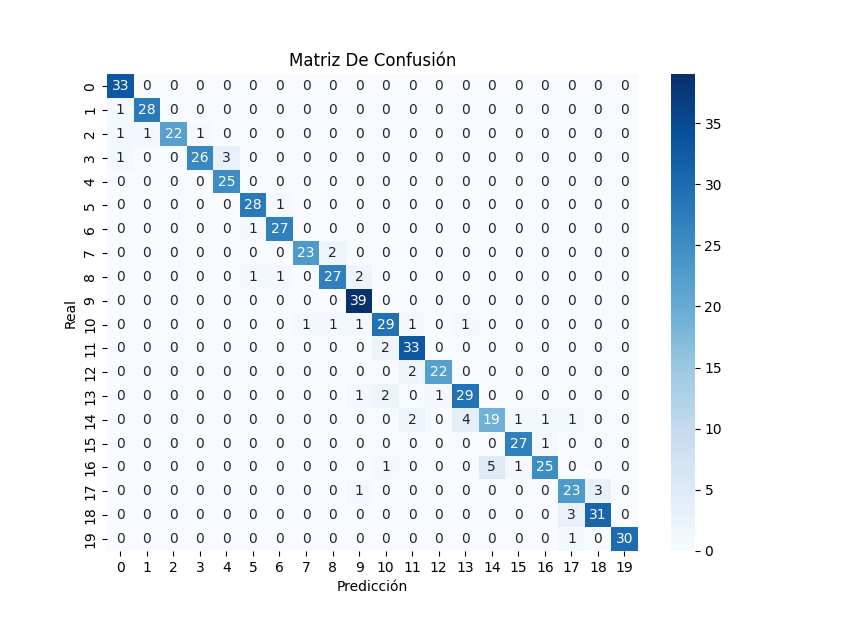
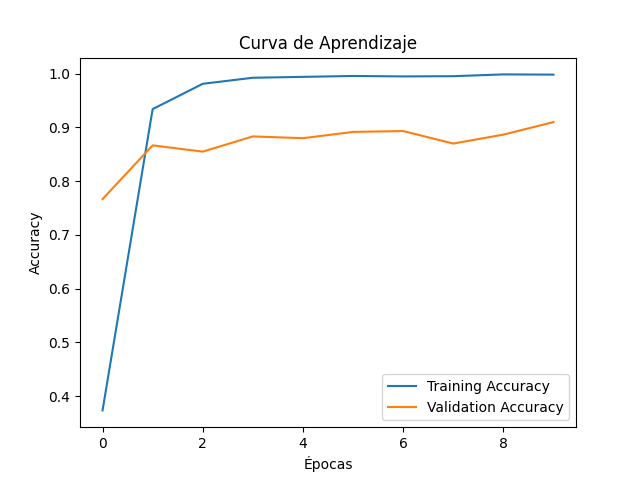


| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 5  BatchSize: 32  Time x Epoch: 160s  Total time: 13,33min  num Imagenes: 2000 RGB  resolución: 225alto x 400ancho  Maíces Con Lentejas  num Clases: 20 (100 fts x 20 class)  testAccuracy: 84,50%  Conv64-128: Crashea  10 epoch: 90%  15 epoch sin dropout: 88,75% |
| --- | --- |



No tuve resultados más relevantes así que a partir de esto probé entrenar maíces solos y maíces con lentejas.

| conv2d, 32f, 3x3, relu  maxPooling 2x2  conv2d 64f, 3x3, relu  maxPooling 2x2  Flatten  Dense 1024n, relu  Dense (numClases)n softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 245s  Total time: 40,83min  num Imagenes: 3000 RGB  resolución: 225alto x 400ancho  Maíces Solos y Con Lentejas  num Clases: 20 (150 fts x 20 class)  testAccuracy: 91%  Conv16-32 5 epoch: 89.50%  Conv16-32-64 5 epoch: 82% |
| --- | --- |



***19/06/2024***

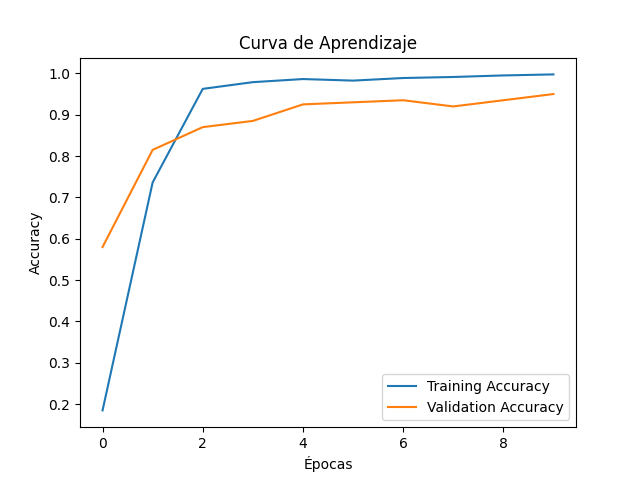
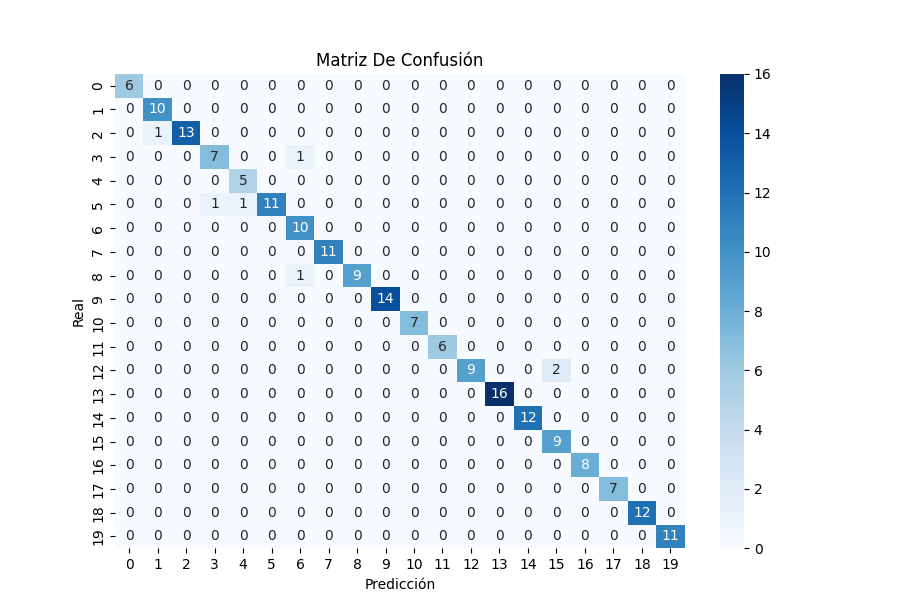
Se empezó a hacer el github con enfoque en secciones como dataset, arquitectura y resultados.

También el informe y diapositivas para la presentación.

Entrene la mejor arquitectura con todos los datasets por separado y sus combinaciones.

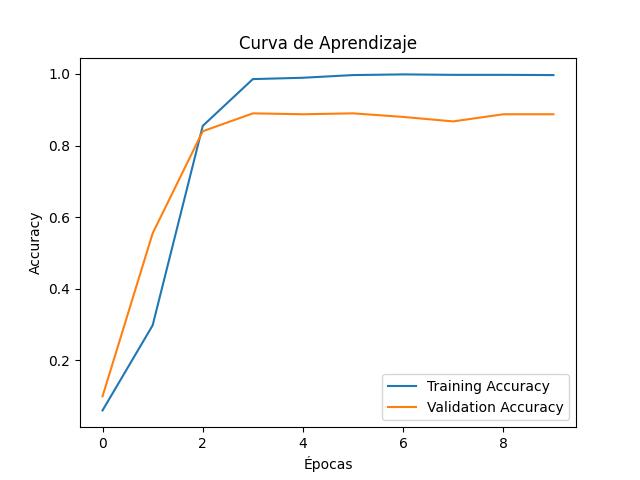
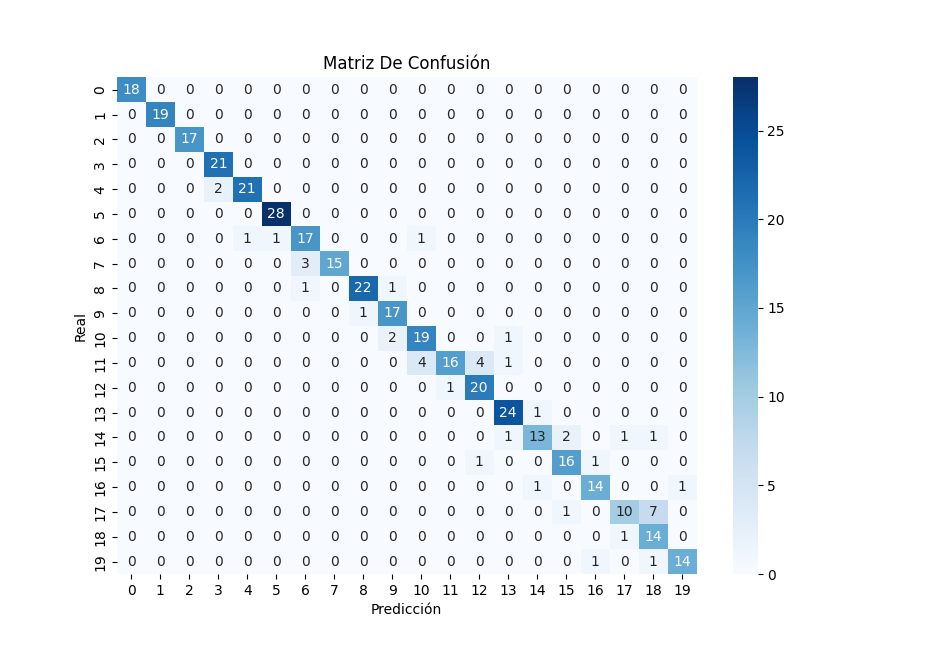
**MAÍCES SOLOS**

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 78s  Total time: 13min  num Imagenes: 1000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (50 fts x 20 class)  testAccuracy: 95,00% |
| --- | --- |

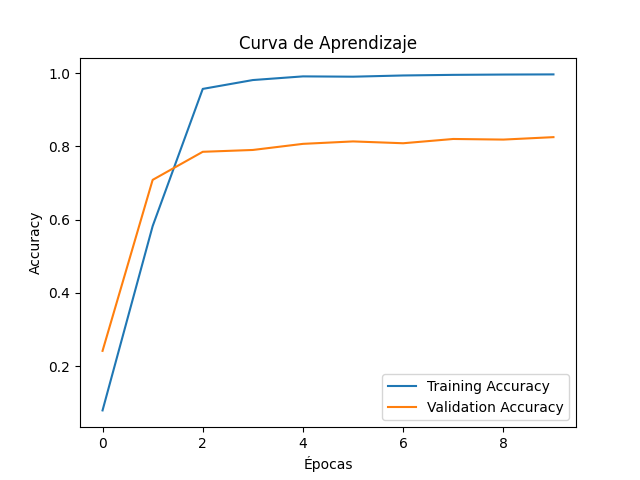
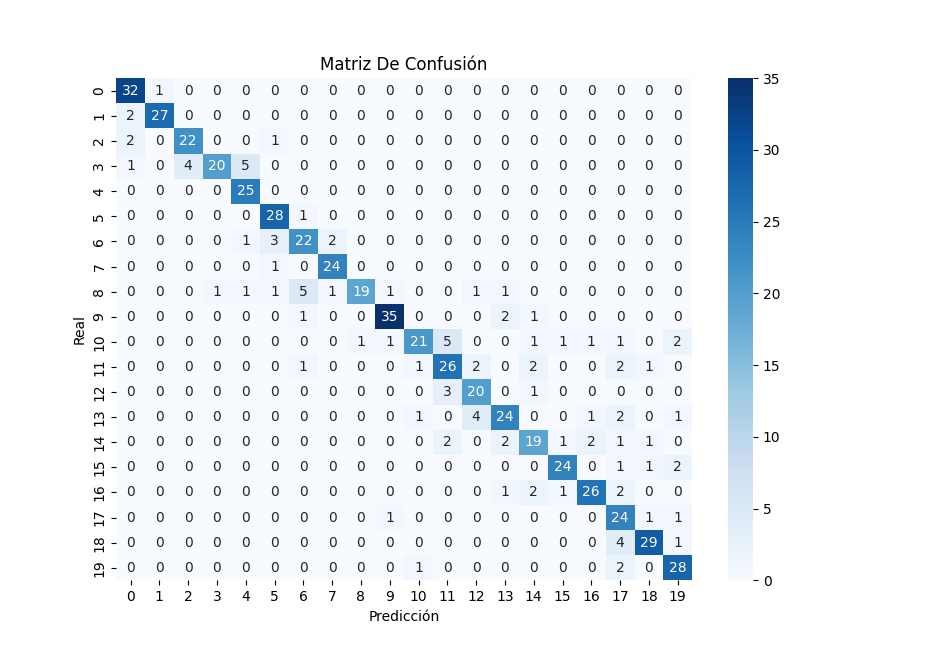
**MAÍCES CON LENTEJAS**

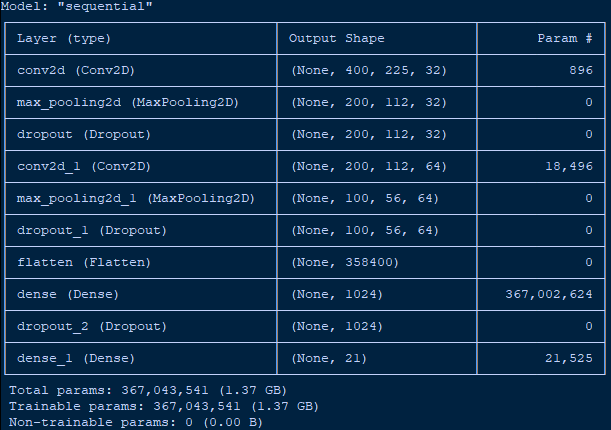
| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 165s  Total time: 27,70min  num Imagenes: 2000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (100 fts x 20 class)  testAccuracy: 88,75%  Variando filtros con 2 y 3 capas no mejora  Variando maxpooling no mejora  Variando dense no mejora  Quitando dropouts: 90% |
| --- | --- |

**MAÍCES SOLOS Y MAÍCES CON LENTEJAS**

| conv2d, 32f, 3x3, relu  maxPooling 2x2  Dropout 0.25  conv2d 64f, 3x3, relu  maxPooling 2x2  Dropout 0.25  Flatten  Dense 1024n, relu  Dropout 0.15  Dense (numClases)n softmax | Epochs: 10  BatchSize: 32  Time x Epoch: 262s  Total time: 43,66min  num Imagenes: 3000 RGB  resolución: 225alto x 400ancho  num Clases: 20 (150 fts x 20 class)  testAccuracy: 82,50% |
| --- | --- |



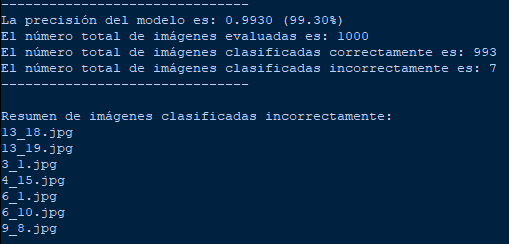
**Se observó que cuando se cambiaban los datasets(maíces solos - maíces con lentejas - maíz lentejas y arroz) y se variaron diferentes arquitecturas que la definida la precisión no mejoraba a menos que se quitaran los dropouts.**

***24/06/2024***

Revise el código de predicción específicamente la parte del cálculo de precisión ya que daba mal los resultados y era difícil de entender así que lo corregí además de ordenar el google drive.

La fórmula que se usa sería el número de predicciones correctas dividido por el número total de predicciones.

También se realizó el [**informe**](https://docs.google.com/document/d/19BybXDD3KyFsC5T4OOszTSbfBJgsMaUO9oh9NKn78Kw/edit#heading=h.kn331ggmgt9q).



***26/06/2024***

**¿QUÉ APRENDE LA RED? ¿CÓMO LO APRENDIÓ? ¿CUANTO APRENDIÓ?**

Con la mejor arquitectura definida (Que fue la mejor para solo maíces) se realizan las diferentes precisiones alcanzadas diferenciando el dataset con el que se entrenó y el dataset con el que se testeo en predicciones (Cada uno por separado o combinaciones de los mismos).

| **DATASET DE ENTRENAMIENTO** | MAÍCES\_SOLOS: 96.50% | MAICES\_LENTEJAS: 90% | MAÍCES\_LENTEJAS\_ARROZ: 98.97% | LOS 3 JUNTOS: 94.85% | MAÍCES SOLOS + MAÍCES LENTEJAS: 90.20% | MAÍCES SOLOS + MAÍCES LENTEJAS ARROZ: 96.94% | MAÍCES LENTEJAS + MAÍCES LENTEJAS ARROZ: 94.02% |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DATASET TESTEO PREDICCIÓN**  **(TODO EL DATASET)** | MAÍCES\_SOLOS: 99.30%  MAICES\_LENTEJAS: 11.95%  MAÍCES\_LENTEJAS\_ARROZ: 0% | MAÍCES\_SOLOS: 6.80%  MAICES\_LENTEJAS: 97.95%  MAÍCES\_LENTEJAS\_ARROZ: 0% | MAÍCES\_SOLOS: 2.70%  MAICES\_LENTEJAS: 4.35%  MAÍCES\_LENTEJAS\_ARROZ: 24.98% | MAÍCES\_SOLOS: 98.00%  MAICES\_LENTEJAS: 98.20%  MAÍCES\_LENTEJAS\_ARROZ: 24.94% | MAÍCES\_SOLOS: 97.90%  MAICES\_LENTEJAS: 97.85%  MAÍCES\_LENTEJAS\_ARROZ: 0.00%  MAÍCES SOLOS + MAÍCES LENTEJAS: 97.87% | MAÍCES\_SOLOS: 98.70%  MAICES\_LENTEJAS: 7.55%  MAÍCES\_LENTEJAS\_ARROZ: 24.90% | MAÍCES\_SOLOS: 5.80%  MAICES\_LENTEJAS: 97.75%  MAÍCES\_LENTEJAS\_ARROZ: 24.94% |

***27/06/2024***

Mientras espero que Ivan me suba las fotos del dataset que falta me dedico a mi otra tarea que sería seguir con el armado del [**GITHUB**](https://github.com/Leandrituw/Contando-Objetos-Con-Redes-Neuronales-Convolucionales?tab=readme-ov-file)**.**

***28/06/2024***

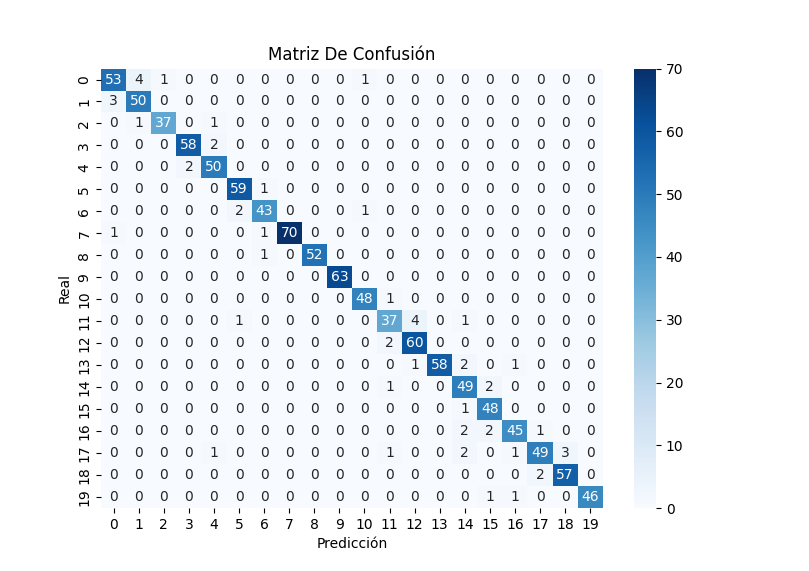
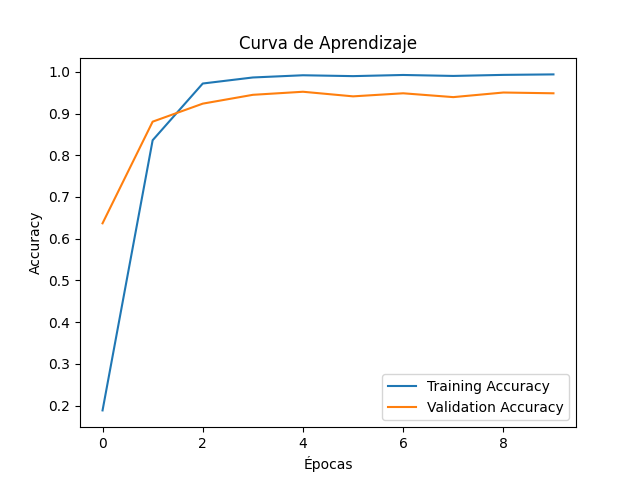
**MEJOR ARQUITECTURA - 10 EPOCH - 80% 20% - TIEMPO: 30/40 MIN APROX C/U**

| **DATASET DE ENTRENAMIENTO** | **DATASET DE TESTEO PREDICCIÓN (TODO)** |
| --- | --- |
| MAÍCES\_SOLOS: 96.50% | MAÍCES\_SOLOS: 99.30%  MAICES\_LENTEJAS: 11.95%  MAÍCES\_LENTEJAS\_ARROZ: 0% |

| **DATASET DE ENTRENAMIENTO** | **DATASET DE TESTEO PREDICCIÓN (TODO)** |
| --- | --- |
| MAICES\_LENTEJAS: 90% | MAÍCES\_SOLOS: 6.80%  MAICES\_LENTEJAS: 97.95%  MAÍCES\_LENTEJAS\_ARROZ: 0% |

| **DATASET DE ENTRENAMIENTO** | **DATASET DE TESTEO PREDICCIÓN (TODO)** |
| --- | --- |
| MAÍCES\_LENTEJAS\_ARROZ: 98.97% | MAÍCES\_SOLOS: 2.70%  MAICES\_LENTEJAS: 4.35%  MAÍCES\_LENTEJAS\_ARROZ: 24.98% |

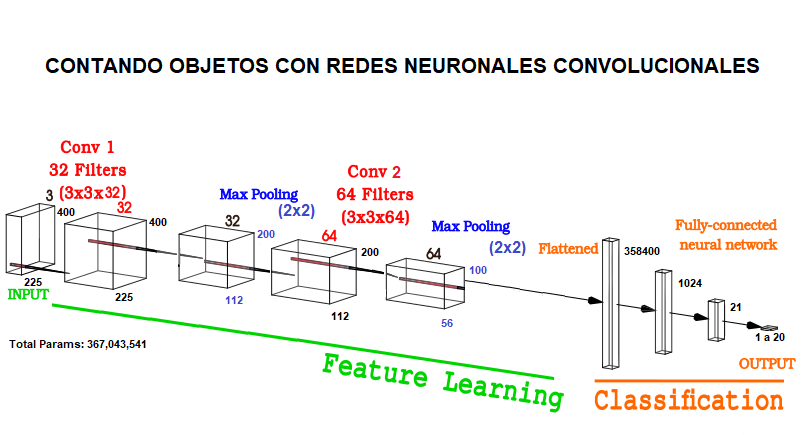
| **DATASET DE ENTRENAMIENTO** | **DATASET DE TESTEO PREDICCIÓN (TODO)** |
| --- | --- |
| LOS 3 JUNTOS: 94.85% | MAÍCES\_SOLOS: 98.00%  MAICES\_LENTEJAS: 98.20%  MAÍCES\_LENTEJAS\_ARROZ: 24.94% |



| **DATASET DE ENTRENAMIENTO** | **DATASET DE TESTEO PREDICCIÓN (TODO)** |
| --- | --- |
| MAÍCES SOLOS + MAÍCES LENTEJAS: 90.20% | MAÍCES\_SOLOS: 97.90%  MAICES\_LENTEJAS: 97.85%  MAÍCES\_LENTEJAS\_ARROZ: 0.00%  MAÍCES SOLOS + MAÍCES LENTEJAS: 97.87% |

***01/07/2024***

Hice las diapositivas para la [**presentación**](https://docs.google.com/presentation/d/1d9TCMheFcDVQgM9nQMwJBvkjVjrUsbma/edit#slide=id.p1) y realice algunas imágenes como el esquema de red final entre otras necesarias tanto para github como para las diapositivas.



**CONCEPTOS/TEORÍA**

**¿QUE ES UNA RED NEURONAL?**

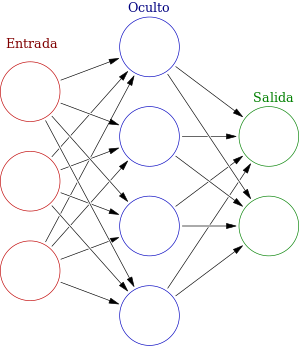
Conocemos como “red neuronal” a un modelo computacional inspirado en el funcionamiento del cerebro humano.

**¿Qué función cumple?**

Procesar información de manera similar a como lo hacen las neuronas en el sistema nervioso biológico. Las redes neuronales artificiales son una parte fundamental del campo de la inteligencia artificial y el aprendizaje automático.

**¿Cómo se estructura y aprende?**

Una red neuronal artificial consta de varias capas de nodos o neuronas interconectadas entre sí. Cada neurona realiza operaciones simples, como la suma ponderada de las entradas seguida de una función de activación que determina la salida de la neurona. Estas conexiones entre neuronas se representan mediante pesos que se ajustan durante el entrenamiento de la red. Las redes neuronales se utilizan para una amplia variedad de aplicaciones, como reconocimiento de patrones, procesamiento de lenguaje natural, visión por computadora, sistemas de recomendación, análisis de datos, entre otros.



**Librerías de Python a utilizar:**

1. TensorFlow
2. NumPy
3. Keras
4. Matplotlib
5. Open-CV
6. Pillow
7. Sklearn
8. Time

**¿QUÉ SON LOS PARADIGMAS DE APRENDIZAJE?**

Cuando hablamos de "paradigmas de aprendizaje" nos referimos a enfoques, estrategias o marcos conceptuales que guían la forma en que se diseña, se entrena y se evalúa un modelo de aprendizaje automático. Estos paradigmas representan diferentes filosofías o enfoques teóricos sobre cómo los modelos de aprendizaje automático deben adquirir y utilizar el conocimiento a partir de los datos.

* Aprendizaje supervisado: Con este tipo de aprendizaje, el proceso de generación del conocimiento se realiza a través de ciertos ejemplos.

Si nuestro input es la foto de un gato, queremos enseñarle que el output debería ser “gato”. Si nuestro input es la foto de un perro, queremos enseñarle que el output debería ser “perro”. Tras darle muchos ejemplos, el algoritmo será capaz de reconocer la etiqueta (output) de fotos que aún no le mostramos.

* Aprendizaje no supervisado: Recopila información sin necesidad de conocer ninguna variable, ya que explora incluso los datos que aún no están etiquetados.
* Aprendizaje reforzado: su objetivo es generar modelos que incrementen el rendimiento y utiliza como referencia la recompensa que se generará con cada interacción que se haya realizado.

**¿QUÉ ES EL OVERFITTING?**

[**¿Qué es el Overfitting?**](https://www.ibm.com/es-es/topics/overfitting)

[**Overfitting - Underfitting y Cómo Solucionarlo**](https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/#:~:text=Tal%20vez%20se%20pueda%20traducir,conocimiento%20que%20pretendemos%20que%20adquieran)

El sobreajuste o overfitting es un comportamiento de aprendizaje automático no deseado que se produce cuando el modelo de aprendizaje automático proporciona predicciones precisas para los datos de entrenamiento, pero no para los datos nuevos.

**¿PORQUE LA PRECISIÓN NO MEJORA SI ESTOY USANDO UN MODELO CARGADO?**

1. **Sobreajuste inicial**: Si el modelo ya estaba sobre ajustado antes de cargarlo, continuar entrenando podría no mejorar la precisión y, de hecho, podría empeorar. En este caso, es posible que necesites técnicas de regularización o una arquitectura de modelo diferente para mejorar la generalización.
2. **Datos de entrenamiento insuficientes**: Si los datos de entrenamiento son limitados o no representativos del problema en cuestión, el modelo puede no ser capaz de aprender patrones más complejos incluso después de más entrenamiento.
3. **Learning rate demasiado alto**: Un learning rate alto puede hacer que el modelo salte a través del óptimo global y no converge adecuadamente. En este caso, puedes intentar reducir el learning rate.
4. **Arquitectura del modelo**: Es posible que la arquitectura del modelo no sea lo suficientemente compleja para capturar la complejidad de los datos. En este caso, puedes probar con una arquitectura más profunda o cambiar la cantidad de neuronas en las capas.
5. **Preprocesamiento de datos incorrecto**: El preprocesamiento de datos puede tener un gran impacto en el rendimiento del modelo. Asegúrate de que estás realizando un preprocesamiento adecuado, como normalización, manejo de valores faltantes, etc.
6. **Hiper Parámetros inadecuados**: Otros hiper parámetros del modelo, como el batch size, el número de epochs, la función de pérdida, etc., podrían no estar optimizados correctamente.

**CURVAS DE APRENDIZAJE**

La curva de aprendizaje es una representación gráfica de cómo evoluciona el rendimiento de un modelo a medida que se incrementa la cantidad de datos de entrenamiento o el número de iteraciones de entrenamiento (épocas). Aquí hay algunos conceptos clave para entender cómo leer una curva de aprendizaje:

1. **Eje x (horizontal):** En general, el eje x representa el número de iteraciones de entrenamiento o épocas. Cada punto en la curva corresponde a una iteración o época durante el entrenamiento del modelo.
2. **Eje y (vertical):** El eje y suele representar la métrica de evaluación del rendimiento del modelo, como la pérdida (loss) o la precisión (accuracy). Esta métrica se calcula utilizando un conjunto de datos de validación o prueba.
3. **Curva de entrenamiento:** La curva que muestra cómo evoluciona la métrica de evaluación en el conjunto de entrenamiento a medida que avanza el entrenamiento. Idealmente, esta curva debería disminuir (en el caso de la pérdida) o aumentar (en el caso de la precisión) a medida que el modelo aprende de los datos.
4. **Curva de validación:** La curva que muestra cómo evoluciona la métrica de evaluación en el conjunto de validación (o prueba) a medida que avanza el entrenamiento. Esta curva proporciona información sobre cómo el modelo generaliza datos no vistos. Idealmente, la curva de validación debería seguir la curva de entrenamiento de cerca. Una brecha significativa entre estas dos curvas podría indicar sobreajuste (overfitting) o subajuste (underfitting) del modelo.

En resumen, una curva de aprendizaje bien interpretada puede proporcionar información valiosa sobre cómo está aprendiendo un modelo y cómo generaliza a datos no vistos. Es importante monitorear tanto la curva de entrenamiento como la de validación para evaluar el rendimiento del modelo de manera efectiva.

**CLASIFICACION Y REGRESION**

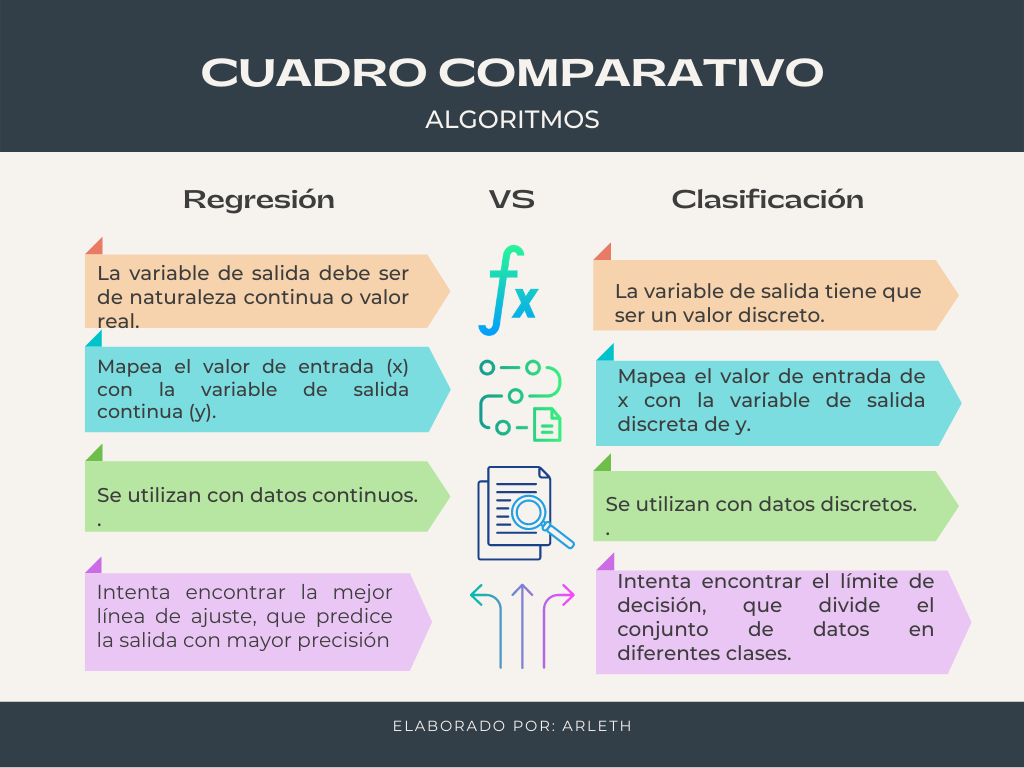
[Diferencias clave entre la Regresión y la Clasificación en Machine Learning](https://www.youtube.com/watch?v=iEqdtiW_UdY&ab_channel=CodigoMaquina)

[DIFERENCIA ENTRE ALGORITMOS DE CLASIFICACIÓN Y REGRESIÓN](https://www.youtube.com/watch?v=u9kchxQAelM&ab_channel=AprendeIAconLigdiGonzalez)

En primer lugar, debemos saber que ambos tipos de problemas se encuentran enmarcados dentro del aprendizaje supervisado del machine learning. En este tipo de aprendizaje, se trabaja con datos etiquetados

Concretamente, la generación del conocimiento se genera mediante ejemplos que permiten predecir, de forma adecuada, el comportamiento de los datos.

A continuación, veremos algunas de las diferencias que existen entre ambas y, aunque algunas técnicas son específicas para cada tipo de algoritmo, otras funcionan para ambos.



**Algoritmos de clasificación**

Los algoritmos de clasificación se utilizan en casos en los que el resultado es un **conjunto infinito de resultados.**

**La clasificación es un algoritmo que ayuda a dividir el conjunto de datos en clases según varios parámetros. Cuando utilizamos un algoritmo de clasificación, le estamos enseñando a un programa en el conjunto de datos de entrenamiento, para que este clasifique los datos en varias categorías según lo que aprendió.**

**Los algoritmos de clasificación encuentran la función de mapeo para mapear la entrada a la salida discreta. Los algoritmos estiman valores discretos, en función de un conjunto particular de variables independientes. Estos se utilizan para la clasificación de correos electrónicos y spam (Si buscamos saber si un correo es o no es spam, el algoritmo de clasificación decide a qué tipo pertenece.) La identificación de células tumorales cancerosas, entre otros.**

A esta metodología también se la conoce por **clasificación binaria.**

**Técnicas utilizadas**

En la actualidad se usan distintas **técnicas de aprendizaje automático para problemas de clasificación.** Entre ellos encontramos:

* La regresión logística
* Redes neuronales y aprendizaje profundo
* **Clasificación del árbol de decisión:** este tipo divide un conjunto de datos en segmentos en función de variables de características particulares.
* **K-vecinos más cercanos:** este tipo de clasificación identifica los K vecinos más cercanos a un punto de observación determinado. Luego usa K puntos para evaluar las proporciones de cada tipo de variable objetivo y predice la variable objetivo que tiene la relación más alta.
* **Regresión logística:** Predice la probabilidad de que Y se asocia con la variable de entrada X.
* **Naïve Bayes:** Se basa en el teorema de Bayes, que describe cómo se evalúa la probabilidad de un evento en función del conocimiento previo de las condiciones que podrían estar relacionadas con el evento.
* **Clasificación de bosque aleatorio:** el bosque aleatorio procesa muchos árboles de decisión, cada uno de los cuales predice un valor para la probabilidad de la variable objetivo, para llegar al resultado final promediando las probabilidades.
* **Máquinas de vectores de soporte:** este algoritmo emplea clasificadores de vectores de soporte mediante la ampliación del espacio de variables de características mediante el empleo de funciones especiales.

**Algoritmos de regresión**

En el caso de los algoritmos de regresión, podemos decir que se trata de un subcampo del aprendizaje automático supervisado que tiene el fin de crear una metodología para relacionar un cierto número de características y una variable objetivo-continua.

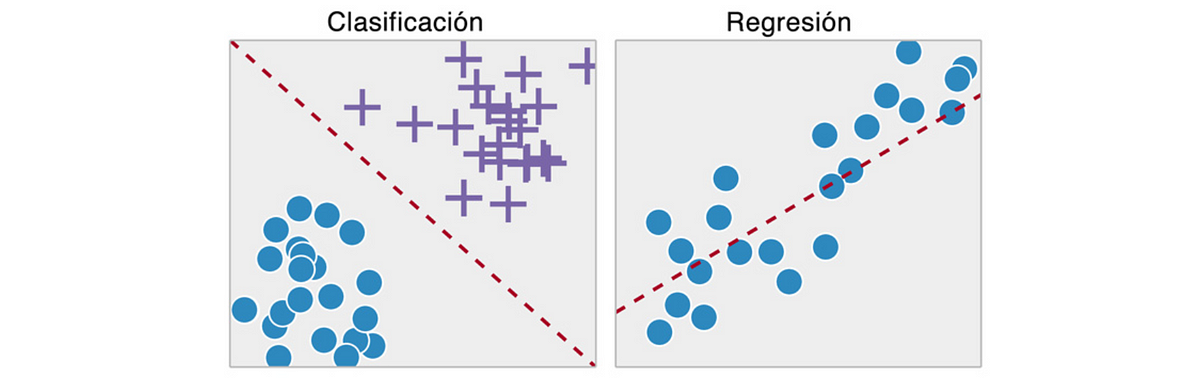
Algunos de los ejemplos más claros de los algoritmos de regresión son la estimación de cuánto tardará una persona en llegar a un destino, la predicción del tiempo que se mantendrá un empleado en una compañía.

La **regresión** encuentra correlaciones entre las variables dependientes e independientes. Por consiguiente, los **algoritmos de regresión** nos pueden ayudar a predecir variables continuas como las tendencias del mercado, los patrones climáticos, entre otros. La función del **algoritmo de regresión** es encontrar la función de mapeo para que podamos mapear la variable de entrada a la variable de salida continua.

**Técnicas utilizadas**

También existen distintas técnicas usadas en los algoritmos de regresión. Entre ellas, podemos encontrar:

* Regresión logística (utiliza datos discretos).
* Deep Learning.
* **Regresión del árbol de decisión:** Se divide el conjunto de datos en subconjuntos más pequeños, para trazar el valor de cualquier punto de datos que se conecte al enunciado del problema.
* **Regresión de Componentes Principales:** Esta técnica de regresión es ampliamente utilizada. Hay muchas variables independientes o existe multicolinealidad en sus datos.
* **Regresión polinomial:** Este tipo ajusta una ecuación no lineal usando las funciones polinómicas de una variable independiente.
* **Regresión de bosque aleatorio:** Esta utiliza múltiples árboles de decisión para predecir la salida, en donde los puntos de datos aleatorios se eligen del conjunto de datos dado y se utilizan para construir un árbol de decisión a través de este algoritmo.
* **Regresión lineal simple:** este tipo es la forma menos complicada de regresión, donde la variable dependiente es continua.
* **Regresión de vectores de soporte:** Utiliza funciones kernel no lineales, para encontrar una solución óptima para modelos no lineales.



**Conclusión**

Finalmente, podemos decir que los **algoritmos de** **regresión** y **clasificación** son **algoritmos del machine learning**. Estos dos **algoritmos** tienen la similitud que se utilizan para la predicción en el **aprendizaje automático** con el uso de conjuntos de datos etiquetados. La principal diferencia entre los **algoritmos de regresión** y **clasificación** es que los **algoritmos de regresión** se usan para predecir valores continuos como precio, salarios, etc. y los **algoritmos de clasificación** se usan para clasificar valores discretos como, verdadero o falso, spam o no spam, entre otros.

**CIFAR-10/100**

[Cómo Entrenar Una Red Automáticamente en CIFAR-10 y CIFAR-100 con Autokeras](https://www.youtube.com/watch?v=l84WLnmqjqg&ab_channel=DataSmarts)

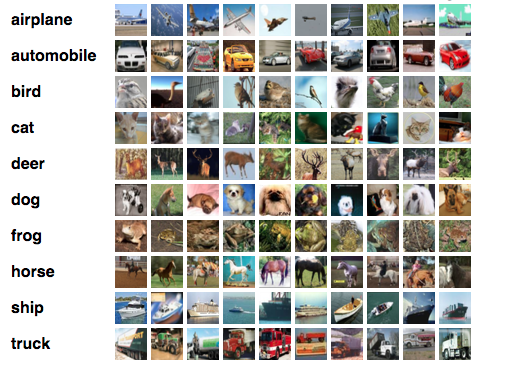
[Cómo Entrenar Una Red Neuronal en CIFAR-10 con Keras Desde Cero](https://www.youtube.com/watch?v=gGfhWi64A4k&ab_channel=DataSmarts)

Toda la información necesaria en el siguiente artículo:

[**Que Es CIFAR-10**](https://datasmarts.net/es/que-es-cifar-10/)

[**Como Entrenar Una Red Con CIFAR-10**](https://datasmarts.net/es/como-entrenar-una-red-neuronal-en-cifar-10-con-keras/)

[**Código En GitHub De Red Con CIFAR-10**](https://gist.github.com/eblancoh/d379d92a3680360857581d8937ef114b)



Cómo subir nuestras propias fotos a python para entrenar nuestra red neuronal?

[(pt. 1/2) Cómo hacer un clasificador de imagenes desde cero con Tensorflow](https://www.youtube.com/watch?v=EAqb20_4Rdg&ab_channel=AMPTech)

[(pt. 2/2) Cómo hacer un clasificador de imagenes desde cero con Tensorflow](https://www.youtube.com/watch?v=FWz0N4FFL0U&ab_channel=AMPTech)

[Ep. 9 Cómo guardar un modelo entrenado](https://www.youtube.com/watch?v=5X3xWlJ2Ozw&t=145s&ab_channel=AMPTech)

1. **¿Cómo trabajan las redes neuronales con imágenes?**

Trabajan las imágenes como matrices de píxeles.

Aplican filtros para extraer características específicas (bordes, patrones, texturas).

1. **¿Qué es el CIFAR-10?**

**CIFAR10** es un dataset ya incluido en la librería tensorflow. Consiste en 60,000 imágenes de 32x32 píxeles en 10 clases diferentes, como aviones, automóviles, pájaros, gatos, etc. 50,000 de las imágenes son para train y 10,000 para test.

**Estructura De CIFAR10:** Cada imagen está etiquetada con una de las 10 categorías y se usa para entrenar algoritmos de aprendizaje automático, especialmente redes neuronales convolucionales (CNN).

**Objetivo De CIFAR10:** El propósito de CIFAR-10 es proporcionar un conjunto de datos desafiante pero manejable para tareas de clasificación de imágenes, permitiendo a los investigadores y desarrolladores probar y mejorar sus modelos de redes neuronales.

**Entrenamiento Con CIFAR10:** Las redes neuronales, como las CNN, aprenden a reconocer patrones visuales en las imágenes de CIFAR-10 a través de capas convolucionales que extraen características como bordes, formas y texturas, seguidas de capas de agrupación y capas densas para la clasificación final.

**Evaluación De Modelos:** Después del entrenamiento, se evalúan los modelos utilizando un conjunto de datos de prueba separado de las imágenes de CIFAR-10 que no se usaron durante el entrenamiento, para medir su capacidad para generalizar y clasificar imágenes nuevas con precisión.

1. **¿Cómo están organizadas las imágenes (inputs)?**

En el conjunto de datos CIFAR-10, las imágenes están organizadas en tensores, que son estructuras de datos multidimensionales similares a matrices. En particular, las imágenes en CIFAR-10 son representadas como tensores tridimensionales.

Cada imagen en CIFAR-10 tiene las siguientes dimensiones:

- Altura de la imagen (número de píxeles en la dimensión vertical).

- Ancho de la imagen (número de píxeles en la dimensión horizontal).

- Canales de color (generalmente 3 canales para rojo, verde y azul en imágenes a color).

Por lo tanto, cada imagen en CIFAR-10 es un tensor tridimensional con la forma `(altura, ancho, canales)`.

Al cargar los datos de CIFAR-10 en Python utilizando `keras.datasets.cifar10`, obtenemos tensores que representan las imágenes de entrenamiento y prueba. Por ejemplo:

**- `train\_images` es un tensor tridimensional que contiene todas las imágenes de training.**

**- `test\_images` es un tensor tridimensional que contiene todas las imágenes de testing**

# Suponiendo que ya cargamos los datos de CIFAR-10 y tienes los tensores train\_images y test\_images

# Acceder a la primera imagen de entrenamiento

primera\_imagen\_entrenamiento = train\_images[0]

# Acceder al píxel en la posición (0, 0) de la primera imagen de entrenamiento

primer\_pixel = primera\_imagen\_entrenamiento[0, 0]

# Acceder a la imagen número 500 de prueba

imagen\_prueba\_500 = test\_images[500]

# Acceder al canal rojo (primer canal) de la imagen de prueba número 500

canal\_rojo = imagen\_prueba\_500[:, :, 0]

**.KERAS VS .H5 ¿COMO CONVIENE GUARDAR NUESTRA RED?**

Al guardar un modelo de red neuronal en Python, principalmente a través de librerías como TensorFlow o Keras, puedes encontrar dos formatos comunes: .h5 y .keras.

1. **.H5 (HDF5 Format):**
   * Este formato es un contenedor de archivos que permite almacenar y organizar grandes cantidades de datos de forma eficiente.
   * Es compatible con una variedad de librerías y herramientas en Python, lo que significa que puedes cargar modelos .h5 en diferentes entornos y aplicaciones.
   * Es muy utilizado en la comunidad de Machine Learning y Deep Learning debido a su eficiencia y versatilidad.
   * Ventajas: Ampliamente compatible, puede ser utilizado con diferentes librerías y herramientas. Fácil de cargar y guardar.
   * Desventajas: Puede ser menos eficiente en términos de almacenamiento comparado con otros formatos más específicos.
2. **.keras (Keras Format):**
   * Este formato es específico de Keras, lo que significa que es optimizado para guardar modelos de Keras y puede ofrecer algunas ventajas específicas en términos de rendimiento o funcionalidades de Keras.
   * Algunas funciones específicas de Keras, como las personalizaciones del modelo o las capas personalizadas, pueden ser mejor soportadas por este formato.
   * Ventajas: Específico de Keras, lo que puede ofrecer algunas optimizaciones específicas para modelos de Keras. Puede ser preferible si estás trabajando exclusivamente con Keras.
   * Desventajas: Puede ser menos compatible con otras librerías o herramientas fuera del ecosistema de Keras. Puede no ser la mejor opción si planeas utilizar tu modelo en entornos que no son de Keras.

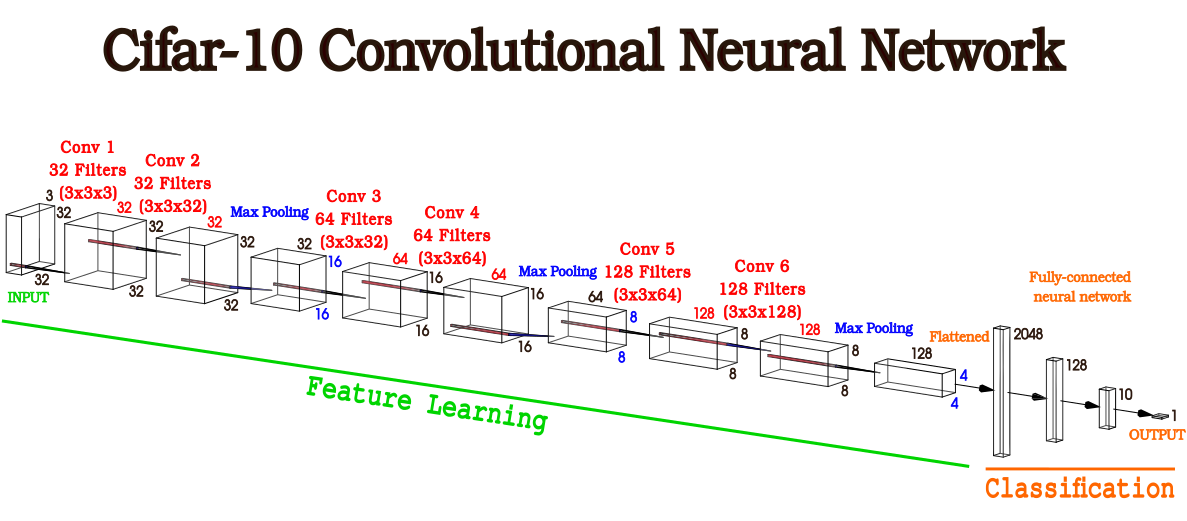
En cuanto a la recomendación de Python para usar .keras, esto puede depender del contexto y de tus necesidades específicas. Si estás trabajando principalmente con Keras y no planeas utilizar tu modelo fuera de ese contexto, .keras podría ser una buena elección. Sin embargo, si necesitas interoperabilidad con otras librerías o herramientas de Machine Learning, o si planeas utilizar tu modelo en diferentes entornos, .h5 podría ser más conveniente debido a su mayor compatibilidad. En última instancia, la elección del formato dependerá de tus necesidades y preferencias específicas.

**REDES NEURONALES CONVOLUCIONALES**

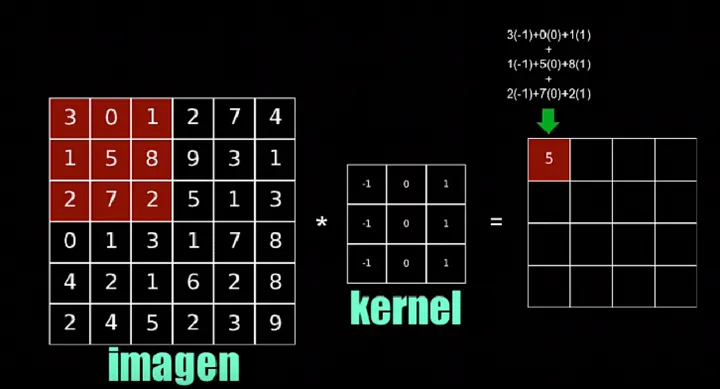
[**Introduccion Redes Convolucionales**](https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8)

[**¿Qué Son Las Redes Neuronales Convolucionales?**](https://www.ibm.com/es-es/topics/convolutional-neural-networks)

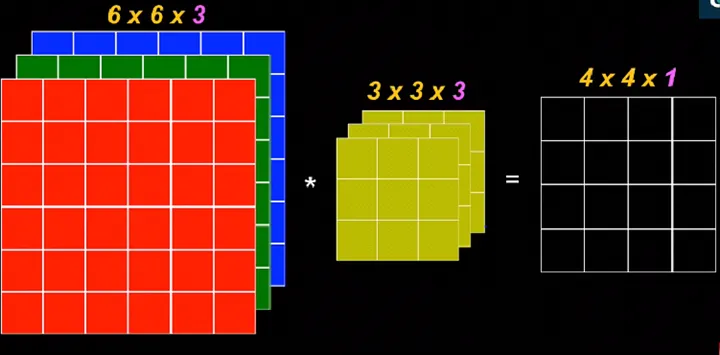
[**Blog Red Convolucional**](https://github.com/UTN-GFA/UTN-GFA.github.io)

****

**ESCALA DE GRISES**

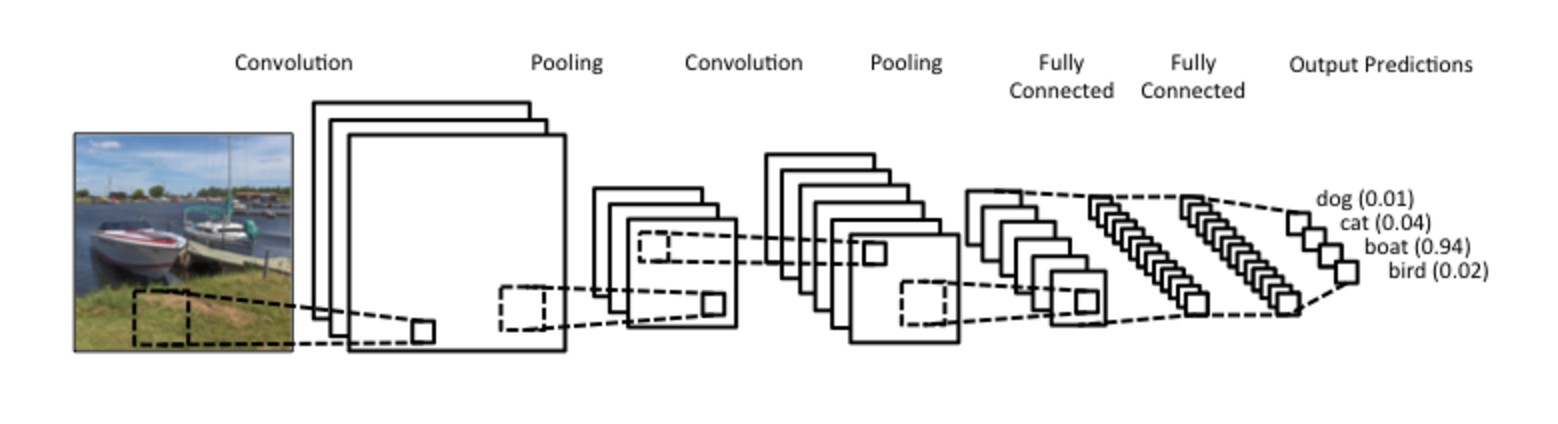
****

**RGB**



**REDES NEURONALES ESTÁNDAR**

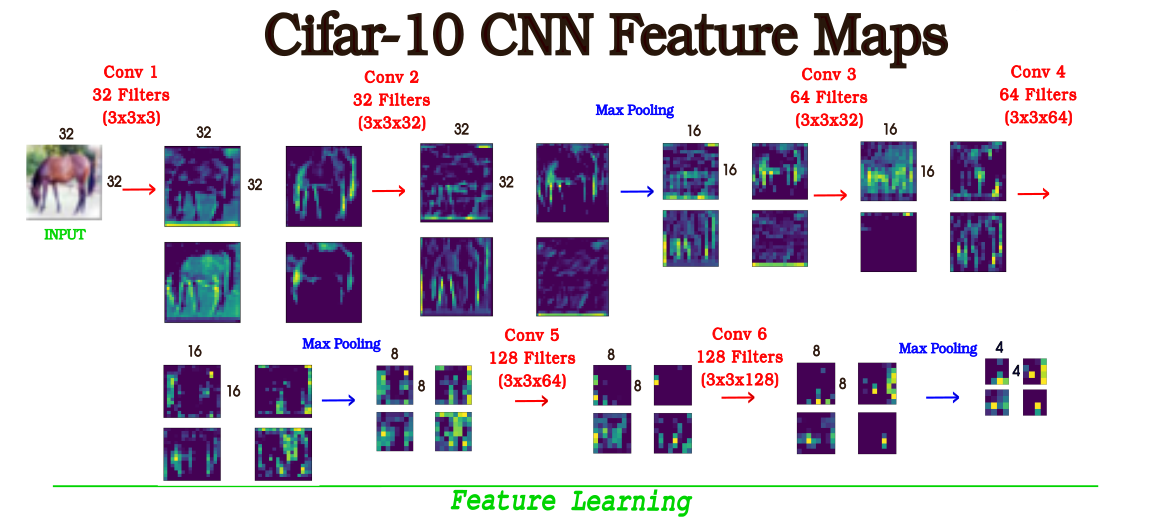
1. **Estructura básica:** Las redes neuronales estándar tienen una estructura de capas densamente conectadas.
2. **Conexiones:** Cada neurona en una capa está conectada a todas las neuronas de la capa anterior y siguiente.
3. **Aplicación:** Se utilizan principalmente para problemas de aprendizaje supervisado, como clasificación y regresión, en datos tabulares o secuenciales.
4. **Desempeño en imágenes:** Pueden tener buen rendimiento en tareas simples de imágenes, pero pueden sufrir de problemas como la falta de invarianza a la traslación y la variabilidad en la posición de los elementos.
5. **Parámetros:** Suelen tener una gran cantidad de parámetros, lo que puede llevar a problemas de sobreajuste en conjuntos de datos pequeños.



**REDES NEURONALES CONVOLUCIONALES (CNN)**

1. **Estructura específica:** Las CNN utilizan capas convolucionales para detectar patrones espaciales en los datos de entrada, como imágenes.
2. **Convoluciones:** Utilizan filtros convolucionales para extraer características locales y aprenden jerarquías de características en capas profundas.
3. **Aplicación:** Son altamente efectivas para el procesamiento de imágenes y vídeos, debido a su capacidad para capturar información espacial y patrones locales.
4. **Invarianza:** Son capaces de aprender invariantes a traslaciones, rotaciones y cambios de escala, lo que las hace robustas para tareas de visión por computadora.
5. **Parámetros compartidos**: Las CNN tienen menos parámetros debido al uso de filtros compartidos, lo que las hace más eficientes y menos propensas al sobreajuste.

**Las redes neuronales convolucionales (CNN) son especialmente poderosas para tareas de visión por computadora, como reconocimiento de imágenes, detección de objetos y segmentación semántica, debido a su capacidad para aprender jerarquías de características espaciales y su invariante a transformaciones. Por otro lado, las redes neuronales estándar son más adecuadas para problemas de naturaleza tabular o secuencial donde la relación entre las características no es especialmente significativa.**



**DIFERENTES TIPOS DE CAPAS**

[**Conv2:**](https://keras.io/api/layers/convolution_layers/convolution2d/)Es la capa convolucional básica que aplica filtros convolucionales a la entrada. Cada filtro convolucional se desliza sobre la entrada para generar mapas de características.

[**MaxPooling2D:**](https://keras.io/api/layers/pooling_layers/max_pooling2d/)Reduce la dimensionalidad de cada mapa de características aplicando una operación de agrupación máxima. Esto ayuda a reducir el número de parámetros y la cantidad de cálculos en la red, lo que puede ayudar a prevenir el sobreajuste y a acelerar el entrenamiento.

**AveragePooling2D:** Similar a MaxPooling2D, pero en lugar de calcular el máximo valor en cada ventana, calcula el promedio de los valores. Puede ser útil en ciertos escenarios, pero MaxPooling2D es más comúnmente utilizado.

**Dense (Fully Connected):** Capa densamente conectada que conecta cada neurona de entrada con cada neurona de salida en capas consecutivas. Esta capa suele seguir a las capas convolucionales y de agrupación para aprender la combinación de características aprendidas por esas capas.

[**Flatten:**](https://keras.io/api/layers/reshaping_layers/flatten/)Transforma los datos de entrada en un vector unidimensional, lo que permite conectar capas convolucionales o de agrupación con capas densamente conectadas.

[**Dropout:**](https://keras.io/api/layers/regularization_layers/dropout/)Esta capa ayuda a prevenir el sobreajuste durante el entrenamiento apagando aleatoriamente un cierto porcentaje de las neuronas de entrada durante cada iteración de entrenamiento.

**BatchNormalization:** Normaliza la activación de cada capa, lo que ayuda a acelerar el entrenamiento y puede mejorar la estabilidad del modelo.

**Activation:** Capa que aplica una función de activación a las salidas de las capas anteriores, como ReLU (Rectified Linear Activation), sigmoid o tanh.

**Concatenate:** Combina las salidas de múltiples capas en una sola salida concatenando los tensores a lo largo de una dimensión específica.

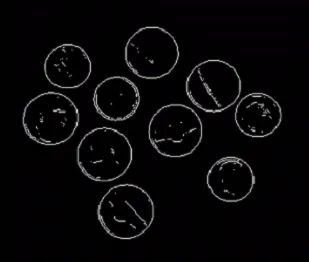
**GlobalAveragePooling2D:** Realiza una operación de agrupación promedio global sobre todo el mapa de características de entrada, reduciendo la dimensionalidad a un solo valor por característica.

Estas son algunas de las capas más comunes utilizadas en las CNN. Dependiendo de la arquitectura específica de la red y la tarea que se esté abordando, es posible que se utilicen otras capas especializadas o variantes de estas capas básicas.



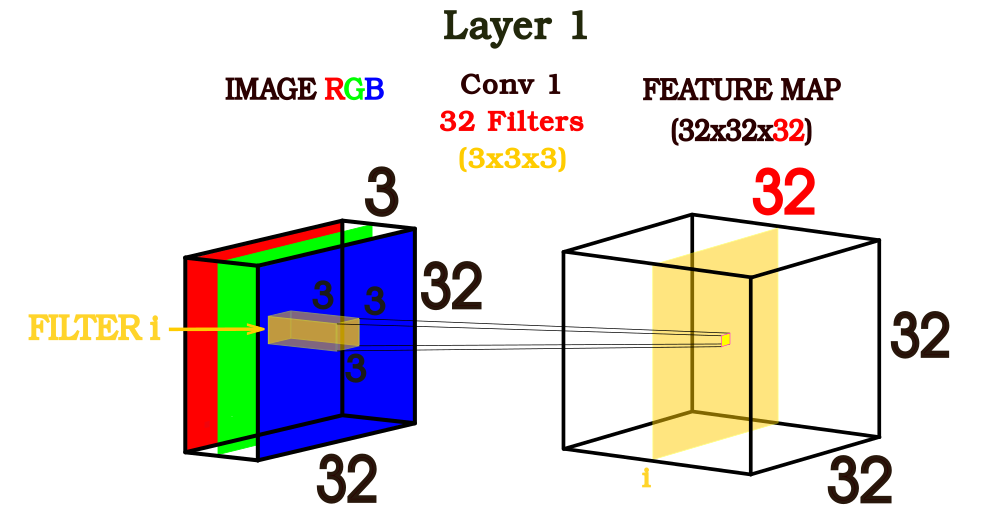
**FILTROS/KERNEL:** Los filtros son las características que la red aprende a reconocer en las imágenes. Un mayor número de filtros permite que la red capture una mayor variedad de características, lo que puede llevar a una representación más rica de los datos. Sin embargo, un exceso de filtros puede aumentar el costo computacional y el riesgo de sobreajuste si no hay suficientes datos para respaldar la complejidad adicional.

**KERNEL**

****

**EPOCHS:** Un epoch es una pasada completa de todos los datos de entrenamiento a través de la red. Entrenar durante más epochs permite que la red aprenda de manera más exhaustiva las características de los datos. Sin embargo, entrenar durante demasiados epochs puede llevar al sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y no generaliza bien a datos nuevos. Es importante monitorear la pérdida en un conjunto de validación para determinar cuándo detener el entrenamiento.

**BATCH SIZE:** El tamaño del lote se refiere al número de ejemplos de entrenamiento que se utilizan en una iteración para actualizar los pesos de la red. Un tamaño de lote más grande puede acelerar el proceso de entrenamiento, ya que la actualización de los pesos se realiza con menos frecuencia, lo que puede aprovechar mejor la eficiencia computacional. Sin embargo, tamaños de lote demasiado grandes pueden requerir más memoria y pueden hacer que el entrenamiento sea menos estable. Además, tamaños de lote más pequeños pueden ayudar a la generalización del modelo al introducir más variabilidad durante el entrenamiento.



**IMAGENET Y ALEXNET**

A partir del blog sobre [**ImageNET y AlexNET**](https://lamaquinaoraculo.com/deep-learning/alexnet/) anote los siguientes conceptos clave.

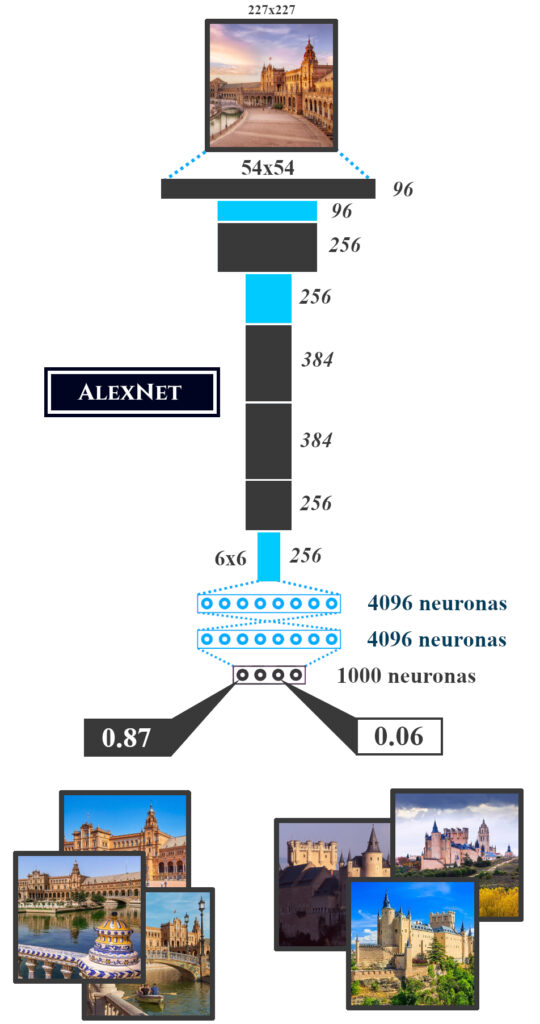
ImageNet es una base de datos de imágenes ampliamente utilizada en el campo del aprendizaje profundo y la visión por computadora. Contiene millones de imágenes etiquetadas en miles de categorías diferentes, lo que la convierte en un recurso invaluable para entrenar y evaluar algoritmos de reconocimiento de imágenes.

Algunas de las características que se encuentran en las imágenes de ImageNet incluyen:

1. **Diversidad de objetos y escenas:** ImageNet contiene imágenes que representan una amplia gama de objetos, animales, personas, paisajes y escenas. Esto incluye desde objetos cotidianos como sillas, automóviles y frutas, hasta animales, edificios, naturaleza, deportes y mucho más.
2. **Variabilidad en la iluminación:** Las imágenes en ImageNet pueden tener diferentes condiciones de iluminación, que van desde escenas bien iluminadas hasta condiciones de poca luz o sombras pronunciadas. Esta variabilidad en la iluminación ayuda a los algoritmos de reconocimiento de imágenes a generalizar mejor a diferentes entornos.
3. **Variabilidad en la orientación y el punto de vista:** Las imágenes en ImageNet pueden mostrar objetos desde diferentes ángulos y puntos de vista. Esto desafía a los algoritmos de visión por computadora a reconocer objetos independientemente de su orientación.
4. **Variabilidad en el fondo y la composición:** Las imágenes pueden tener fondos simples o complejos, y la composición de la imagen puede variar considerablemente. Algunas imágenes pueden tener fondos despejados que destacan claramente el objeto de interés, mientras que otras pueden tener fondos distractivos o confusos.
5. **Variabilidad en la resolución y la calidad:** Las imágenes en ImageNet pueden tener diferentes resoluciones y niveles de calidad. Algunas imágenes pueden ser de alta resolución y alta calidad, mientras que otras pueden ser más pixeleadas o tener artefactos de compresión.

En resumen, las imágenes en ImageNet son diversas en términos de los objetos y escenas que representan, así como en términos de condiciones de iluminación, orientación, composición y calidad. Esta diversidad ayuda a los investigadores y desarrolladores a entrenar y evaluar algoritmos de visión por computadora en una variedad de situaciones del mundo real.

AlexNet, por otro lado, es una arquitectura de red neuronal convolucional (CNN) desarrollada por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton. Fue presentada en 2012 y marcó un hito significativo en el campo del aprendizaje profundo al ganar el concurso ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en ese año, superando ampliamente a los métodos anteriores. AlexNet ayudó a popularizar el uso de CNNs en tareas de visión por computadora y estableció el modelo para muchas arquitecturas de CNN posteriores.

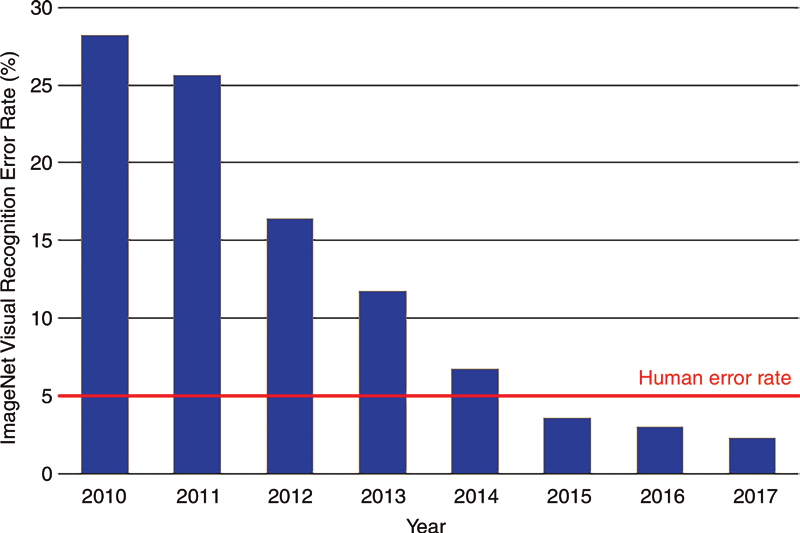


AlexNet es notable por varias características clave que contribuyeron a su éxito y a la revolución en el campo de la visión por computadora:

1. **Arquitectura profunda:** AlexNet fue una de las primeras redes neuronales convolucionales (CNN) profundas en demostrar un rendimiento significativamente mejor que los métodos tradicionales en tareas de clasificación de imágenes. Consiste en ocho capas de aprendizaje profundo, incluyendo cinco capas convolucionales y tres capas totalmente conectadas.
2. **Capas convolucionales y de pooling:** Las capas convolucionales se encargan de extraer características de las imágenes, mientras que las capas de pooling reducen la dimensionalidad de las características, lo que ayuda a controlar el sobreajuste y mejorar la eficiencia computacional.
3. **Funciones de activación no lineales:** AlexNet utiliza la función de activación ReLU (Rectified Linear Unit) en lugar de las funciones sigmoideas o tangentes hiperbólicas más comunes en ese momento. ReLU ayudó a acelerar el entrenamiento de la red y a superar el problema de la desaparición del gradiente.
4. **Regularización:** Para mitigar el sobreajuste, AlexNet utiliza técnicas de regularización como la regularización L2 y el dropout, que consiste en aleatoriamente "apagar" unidades en la red durante el entrenamiento para evitar la coadaptación de los nodos.
5. **Paralelismo en GPUs:** AlexNet fue diseñada para aprovechar al máximo el paralelismo de las GPU (Unidades de Procesamiento Gráfico). Esto permitió un entrenamiento más rápido de la red, lo que fue crucial dada la profundidad de la arquitectura.
6. **Uso de técnicas de aumento de datos:** AlexNet utilizó técnicas de aumento de datos durante el entrenamiento para aumentar la diversidad del conjunto de datos, como la rotación, el recorte aleatorio y el cambio de color. Esto ayudó a mejorar la capacidad de generalización del modelo.

En conjunto, estas características permitieron a AlexNet superar significativamente a los métodos anteriores en la tarea desafiante de clasificación de imágenes en el conjunto de datos ImageNet, estableciendo así un nuevo estándar en el campo de la visión por computadora.

**GANADORES**



1. **2012:** El equipo de Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton de la Universidad de Toronto ganó de manera dominante con su arquitectura AlexNet, marcando un avance significativo en la precisión del reconocimiento de imágenes.
2. **2013:** Los investigadores del laboratorio Visual Geometry Group de la Universidad de Oxford ganaron el desafío con una red neuronal convolucional llamada ZFNet.
3. **2014:** El equipo de Google, liderado por Christian Szegedy y Geoff Hinton, ganó el desafío con una arquitectura llamada Inception (GoogLeNet).
4. **2015:** El equipo de Microsoft Research Asia, liderado por Kaiming He, ganó el desafío con una red neuronal convolucional llamada ResNet, que se destacó por su profundidad.
5. **2016:** El equipo de la Universidad de Oxford, liderado por Kaiming He y Jian Sun, ganó nuevamente utilizando una versión mejorada de la red ResNet.
6. **2017:** El equipo de Google Brain, liderado por Mingxing Tan y Quoc V. Le ganó el desafío con una arquitectura llamada NASNet, desarrollada utilizando métodos de búsqueda automatizada de arquitectura.
7. **2018:** El equipo de Huawei, liderado por Xingyu Gao, ganó el desafío con una arquitectura llamada Squeeze-and-Excitation Networks (SENet).
8. **2019:** El equipo de Google AI, liderado por Mark Sandler, Andrew Howard y Menglong Zhu, ganó el desafío con EfficientNet, una familia de arquitecturas de redes neuronales escalables basadas en un enfoque de optimización compuesto.

A partir de 2020, el ILSVRC fue reemplazado por otros desafíos y competiciones en el campo del aprendizaje profundo y la visión por computadora. Por ejemplo, la comunidad de investigación continúa organizando desafíos como el NeurIPS (Conferencia sobre Sistemas de Procesamiento Neural) y otros eventos académicos y competiciones en línea.

[**CONVOLUCIÓN ATROZ EN CNN**](https://www.analyticsvidhya.com/blog/2023/12/a-comprehensive-guide-on-atrous-convolution-in-cnns/)

La **"Convolución Atroz"** es una **técnica de aumento de datos en aprendizaje profundo que se utiliza para mejorar el rendimiento de los modelos entrenados en conjuntos de datos pequeños o desequilibrados**. La idea principal es aplicar una convolución 2D a las imágenes de entrada para generar nuevas imágenes con variaciones aleatorias pero realistas. Estas variaciones incluyen rotaciones, traslaciones, zooms, cambios de brillo, entre otros, lo que ayuda al modelo a aprender características más robustas y generalizables.

**¿Cómo funciona la Convolución Atroz?**

**Generación de imágenes aumentadas:** Se aplican transformaciones aleatorias a las imágenes de entrenamiento, como rotaciones, traslaciones, cambios de escala, volteos, entre otros. Estas transformaciones se aplican de manera aleatoria en cada época del entrenamiento.

**Variabilidad controlada:** Aunque las transformaciones son aleatorias, se controla la magnitud de estas variaciones para mantener la coherencia visual y evitar generar imágenes completamente irreconocibles.

**Aumento de datos:** El objetivo es crear un conjunto de datos de entrenamiento más grande y diverso, lo que ayuda a prevenir el sobreajuste y mejora la capacidad del modelo para generalizar a datos nuevos y diferentes.

**Mejora de la generalización:** Al exponer al modelo a una variedad más amplia de imágenes, se espera que aprenda características más robustas y relevantes, lo que puede conducir a un mejor rendimiento en datos de prueba.

**¿Cómo se aplica?**

Durante el entrenamiento de la CNN, aplicar transformaciones aleatorias a las imágenes de entrada antes de pasarlas a la red para cada epoch del entrenamiento.

Estas transformaciones pueden incluir rotaciones aleatorias, traslaciones, cambios de escala (zoom), volteos horizontales o verticales, cambios de brillo o contraste, entre otros.

La implementación de estas transformaciones puede depender de la biblioteca que estemos usando. Por ejemplo, en TensorFlow y Keras, se puede trabajar con la función ImageDataGenerator para aplicar fácilmente transformaciones de aumento de datos durante el entrenamiento.

**Un resumen:**

La Convolución Atroz es una técnica efectiva para mejorar la capacidad de generalización de los modelos de aprendizaje profundo al introducir variaciones realistas en los datos de entrenamiento, lo que ayuda al modelo a aprender patrones más robustos y a evitar el sobreajuste.

**CNN FAMOSAS-UTILIDAD-INPUTS-COMPETENCIAS**

1. **LeNet-5**
   * **Para qué sirve:** Fue diseñada originalmente para la clasificación de dígitos escritos a mano en imágenes (usada en la base de datos MNIST).
   * **Tamaño del input:** 32x32 píxeles.
   * **Tamaño del input variable:** No, debe ser fijo.
2. **AlexNet**
   * **Para qué sirve:** Ganó el concurso ImageNet en 2012 y se utiliza para la clasificación de imágenes a gran escala.
   * **Tamaño del input:** 227x227 píxeles.
   * **Tamaño del input variable:** No, debe ser fijo.
3. **VGG16**
   * **Para qué sirve:** Conocida por su simplicidad y profundidad, se utiliza principalmente para la clasificación de imágenes.
   * **Tamaño del input:** 224x224 píxeles.
   * **Tamaño del input variable:** No, debe ser fijo.
4. **ResNet (ResNet-50)**
   * **Para qué sirve:** Introdujo las conexiones residuales y se utiliza para tareas de clasificación y detección de objetos.
   * **Tamaño del input:** 224x224 píxeles.
   * **Tamaño del input variable:** No, debe ser fijo.
5. **Inception (GoogLeNet)**
   * **Para qué sirve:** Ganó el concurso ImageNet en 2014 y se utiliza para la clasificación de imágenes, detección y segmentación.
   * **Tamaño del input:** 224x224 píxeles.
   * **Tamaño del input variable:** No, debe ser fijo.

**Competencias famosas de redes neuronales convolucionales**

1. **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
   * **Descripción:** Una de las competiciones más prestigiosas en el campo del aprendizaje profundo y visión por computadora. Incluye tareas como clasificación de imágenes, detección de objetos y segmentación de escenas.
   * **Relación con CNNs:** Las CNNs se han utilizado extensamente en esta competencia y han sido responsables de muchos avances en el campo.
2. **COCO (Common Objects in Context)**
   * **Descripción:** Una competencia y base de datos orientada a la segmentación y la detección de objetos en imágenes cotidianas.
   * **Relación con CNNs:** Se utilizan CNNs avanzadas para tareas de detección de objetos, segmentación y conteo de instancias de objetos.
3. **PASCAL VOC (Visual Object Classes)**
   * **Descripción:** Una competencia enfocada en la detección y segmentación de objetos en imágenes.
   * **Relación con CNNs:** Las CNNs han sido fundamentales en la mejora del rendimiento en esta competencia.

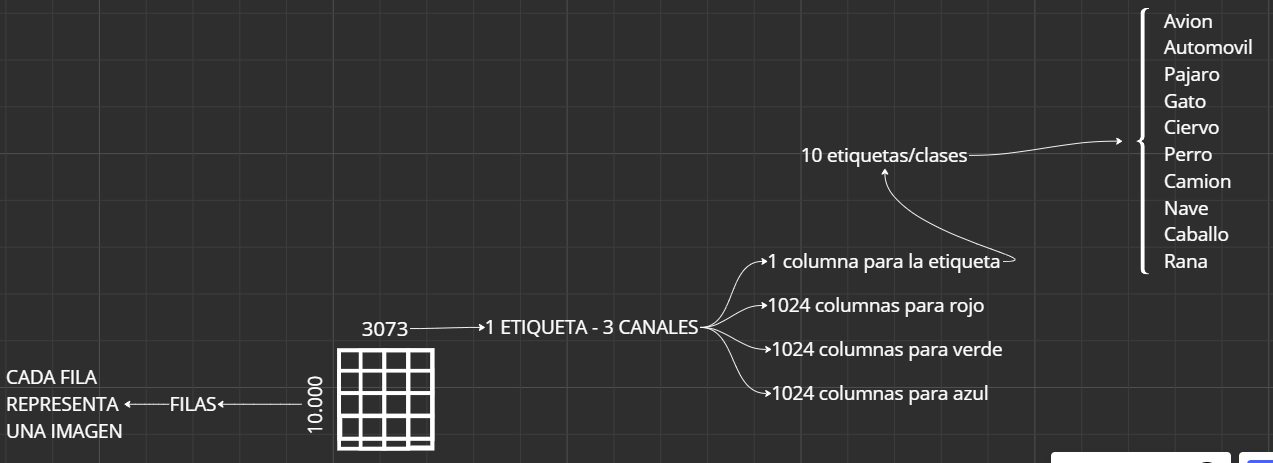
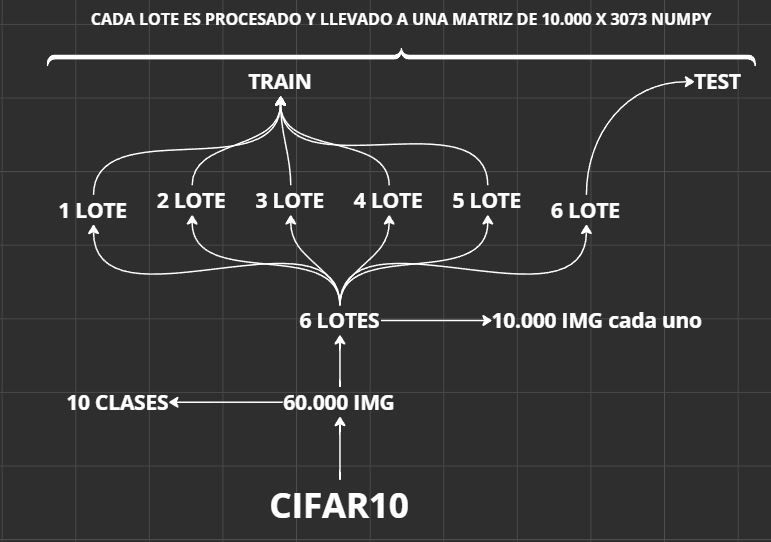
**Redes neuronales convolucionales dedicadas a contar objetos**

Sí, hay modelos y arquitecturas específicas para el conteo de objetos, a menudo utilizando CNNs:

* **YOLO (You Only Look Once)**
  + **Descripción:** Una arquitectura de detección de objetos que puede contar objetos en imágenes y videos en tiempo real.
  + **Tamaño del input:** Variable, aunque normalmente se redimensiona a 416x416 píxeles para procesamiento.
  + **Usos:** Detección y conteo de objetos en diversas aplicaciones, incluyendo vigilancia, vehículos autónomos y análisis de escenas.
* **Counting CNNs (Redes neuronales convolucionales para conteo)**
  + **Descripción:** Redes diseñadas específicamente para el conteo de objetos, como vehículos, células en imágenes biomédicas, personas en multitudes, etc.
  + **Ejemplo:** CSRNet (Convolutional Neural Network for Crowd Counting) se utiliza específicamente para contar personas en imágenes de multitudes.
  + **Tamaño del input:** Variable, aunque a menudo se redimensiona para adaptarse al modelo.

**COMO CREAR UN DATASET SIMILAR A CIFAR-10**

[**Blog De Funcionamiento De CIFAR-10/100**](https://www.cs.toronto.edu/%7Ekriz/cifar.html)



[**Como Crear Un DataSet Similar a CIFAR-10**](https://stackoverflow.com/questions/35032675/how-to-create-dataset-similar-to-cifar-10)

1. **Organizar Las Imágenes**

Primero, las imágenes deben estar organizadas en **carpetas según su categoría**. Por ejemplo:

dataset/

class1/

img1.jpg

img2.jpg

...

class2/

img1.jpg

img2.jpg

...

...

1. **Crear Un Script Para Procesar y Guardar El Dataset**

Escribir un script en Python para procesar estas imágenes, convertirlas en arrays y guardarlas en un formato adecuado (por ejemplo, HDF5 o TFRecord).

1. **Ejemplo De Script En Python**

Una forma facil es usando NumPy y h5py para guardar las imágenes y etiquetas en un archivo HDF5:

import os

import numpy as np

import h5py

from PIL import Image

# Ruta al directorio del dataset

dataset\_dir = 'ruta/a/tu/dataset'

# Tamaño de las imágenes

img\_width, img\_height = 225, 400

# Obtener las clases del dataset

classes = os.listdir(dataset\_dir)

num\_classes = len(classes)

# Inicializar listas para almacenar las imágenes y etiquetas

images = []

labels = []

# Procesar cada clase

for class\_idx, class\_name in enumerate(classes):

class\_dir = os.path.join(dataset\_dir, class\_name)

for img\_name in os.listdir(class\_dir):

img\_path = os.path.join(class\_dir, img\_name)

img = Image.open(img\_path).resize((img\_width, img\_height))

img\_array = np.array(img)

# Asegurarse de que la imagen tiene tres canales (RGB)

if img\_array.shape == (img\_height, img\_width, 3):

images.append(img\_array)

labels.append(class\_idx)

# Convertir listas a arrays de NumPy

images = np.array(images)

labels = np.array(labels)

# Guardar las imágenes y etiquetas en un archivo HDF5

with h5py.File('dataset.h5', 'w') as h5f:

h5f.create\_dataset('images', data=images)

h5f.create\_dataset('labels', data=labels)

```

1. **Cargar El Dataset**

Para cargar el dataset en tus modelos de aprendizaje profundo, puedes escribir un script que lea el archivo HDF5:

import h5py

# Cargar el dataset

with h5py.File('dataset.h5', 'r') as h5f:

images = h5f['images'][:]

labels = h5f['labels'][:]

print(f'Loaded {images.shape[0]} images with shape {images.shape[1:]}')

1. **Dividir El Dataset**

Para dividir tu dataset en conjuntos de entrenamiento y prueba, puedes usar `train\_test\_split` de Scikit-learn:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

print(f'Training set: {X\_train.shape[0]} samples')

print(f'Test set: {X\_test.shape[0]} samples')

```

1. **Guardar Los Conjuntos De Entrenamiento y Prueba Por Separado (Opcional)**

Puedes guardar los conjuntos de entrenamiento y prueba en archivos HDF5 separados si lo deseas:

with h5py.File('train\_dataset.h5', 'w') as h5f:

h5f.create\_dataset('images', data=X\_train)

h5f.create\_dataset('labels', data=y\_train)

with h5py.File('test\_dataset.h5', 'w') as h5f:

h5f.create\_dataset('images', data=X\_test)

h5f.create\_dataset('labels', data=y\_test)

**RESUMEN**

1. Organiza tus imágenes en carpetas según sus categorías.

2. Usa un script en Python para cargar y procesar las imágenes.

3. Guarda las imágenes y etiquetas en un archivo HDF5.

4. Usa el archivo HDF5 para cargar el dataset en tus modelos de aprendizaje profundo.

[**MATRIZ DE CONFUSIÓN**](https://datascientest.com/es/matriz-de-confusion#:~:text=La%20matriz%20de%20confusi%C3%B3n%2C%20tambi%C3%A9n,deben%20ser%20clasificados%20como%20spam)

[**Cómo Interpretarla**](https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico)

La matriz de confusión, también conocida como matriz de error, es un instrumento tecnológico que sirve para calcular el rendimiento sobre un modelo de clasificación definido. De este modo, es posible predecir fácilmente por ejemplo los correos electrónicos que deben ser clasificados como spam.