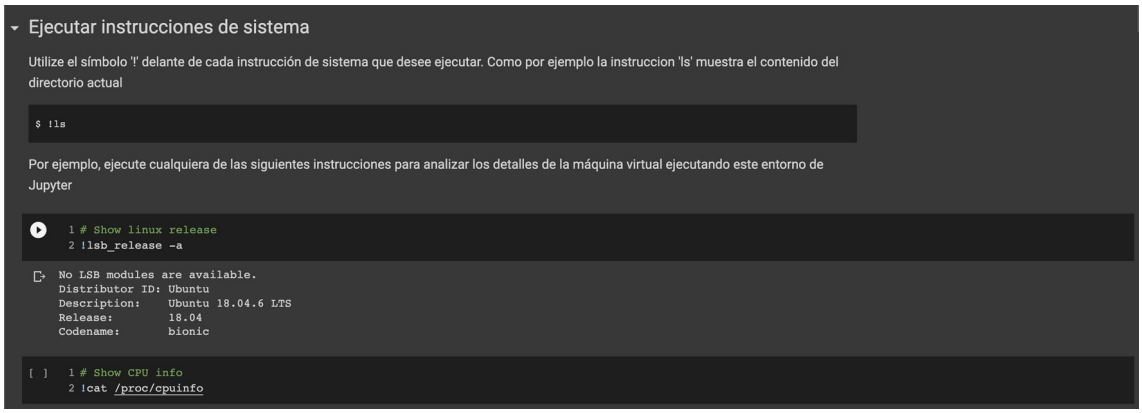
Ejercicio 1 (Notebook: 1.setup_p37.ipynb)

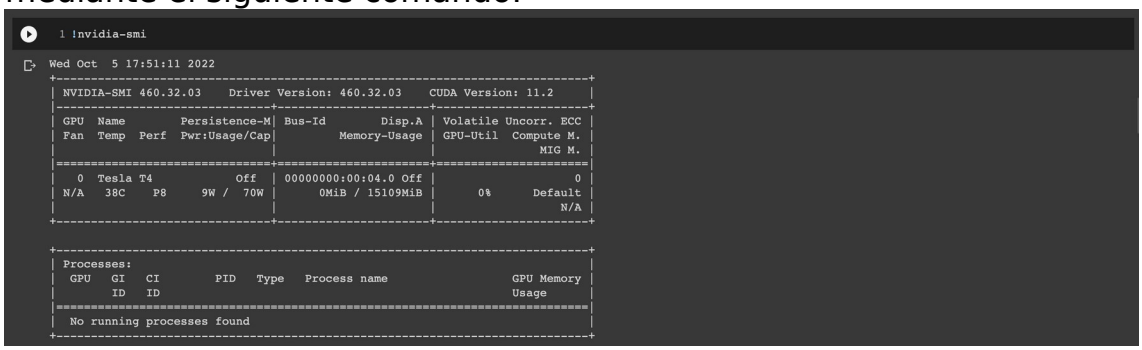
En este notebook se realizan una serie de operaciones que muestran al usuario el HW con el que se encuentra uno cuando accede a la plataforma de Google Colab. También enseña como desde collab se pueden hacer llamadas al SO mediante Bash, utilizando distintos comandos como los siguientes:



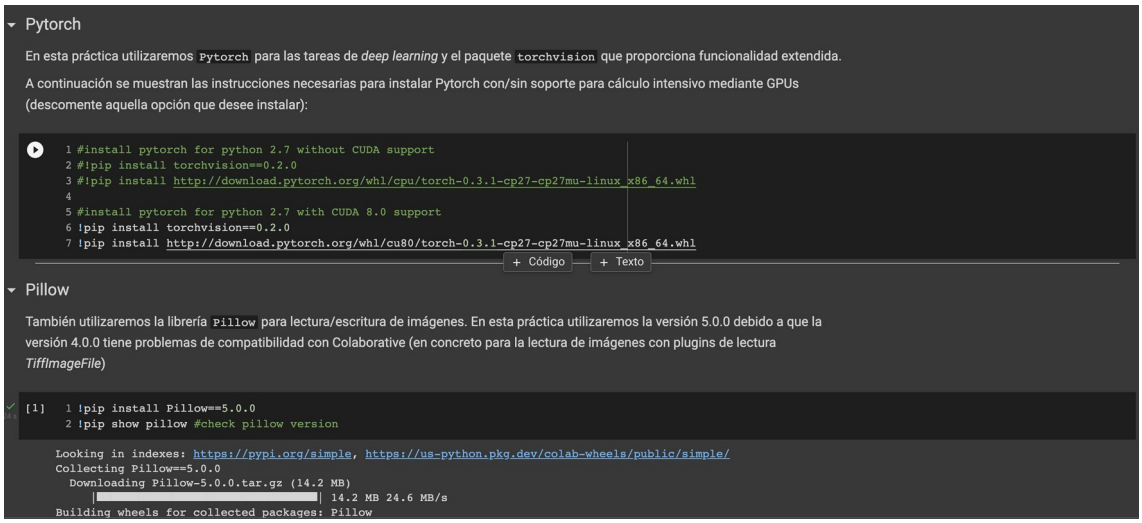
Esta información mostrará tanto los directorios disponibles en el entorno como las especificaciones de la “máquina” que nos asigna Collab cuando nos conectamos a la plataforma:

Para el entrenamiento acelerado de algoritmos de aprendizaje profundo es útil tener GPUs. Estos aceleradores hardware también

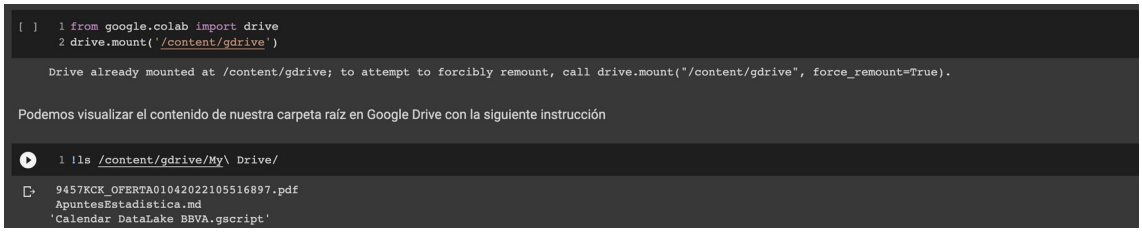
están disponibles en el entorno de ejecución y pueden ser verificados mediante el siguiente comando:



También se realizan una serie de instalaciones de librerías que serán necesarias para poder entrenar a nuestros algoritmos de aprendizaje profundo. Estás librerías serían PyTorch y Pillow:



Por otro lado, también se interactúa con la librería de google, para el manejo de directorios de Drive:



Por último, se realizan una serie de ejecuciones a sentencias en las que se interactúa con el sistema operativo. Estas son, por ejemplo:

Movimiento de directorios:

```
#show current directory
!pwd

#move to a specific directory
import os
os.chdir('/content/gdrive/My Drive/practicalindexacion/')
!pwd
```

```
/content/gdrive/MyDrive/practicalindexacion
/content/gdrive/My Drive/practicalindexacion
```

Generación de scripts “on the run”:

Generación de scripts

Frecuentemente, tras desarrollar una nueva funcionalidad es conveniente utilizar un fichero *.py* para encapsular esta funcionalidad.

En el siguiente código puede ver como se generan el fichero *textproc.py*

```
%%writefile textproc.py

def plural(word):
    if word.endswith('y'):
        return word[:-1] + 'ies'
    elif word[-1] in 'sx' or word[-2:] in ['sh', 'ch']:
        return word + 'es'
    elif word.endswith('an'):
        return word[:-2] + 'en'
    else:
        return word + 's'
```

```
Overwriting textproc.py
```

Ejecución de scripts:

Ejecución de scripts

Tras encapsular la funcionalidad deseada en un fichero *.py*, tenemos dos maneras utilizar dicha funcionalidad

```
#example 1 for importing library
from textproc import plural

#create object
plural('wish')
```

```
'wishes'
```

```
#example 2 for running an external script

%run -i textproc.py #runs script and keeps variables
plural('wish')
```

```
'wishes'
```

Ejercicio 2 (Notebook: 2.dataset_loading_p37.ipynb)

En este dataset se cargan los datos que provienen del dataset CIFAR10. Este dataset tiene las siguientes características:

- 60,000 imágenes a color RGB (con resolución 32x32)
- 10 clases (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)
- Datos están divididos entre entrenamiento/train (50K) y test/validation (10K).

Se convierte el dataset a un formato PyTorch. Este formato tiene dos parámetros clave:

1. **batch_size**: tamaño del grupo de datos que se generan para el entrenamiento y la validación
2. **shuffle**: FLAG que indica si el agrupamiento es aleatorio o secuencial.

Posteriormente, se definen unas funciones que permiten la visualización de datos:

También se presenta una funcionalidad para el tratamiento de datasets no genéricos. En este caso descargaremos el el dataset scene15.

Se volverán a hacer las mismas operaciones, carga en memoria de los datos, transformación al formato PyTorch y la muestra de las imagenes.

Scene15: visualización

De manera similar al caso anterior, procedemos a visualizar los datos cargados en un *batch* aleatorio. Utilizaremos la librería *matplotlib* y *numpy*.

NOTA: si al ejecutar este código obtiene un error del tipo:

```
Image.register_extensions(TiffImageFile.format, [".tif", ".tiff"])
AttributeError: 'module' object has no attribute 'register_extensions'
```

Esto se debe al uso incorrecto de la librería *Pillow*. Para solucionarlo, restablezca el *runtime* (en el menu "Runtime"->"Restart Runtime") y vuelva a ejecutar las instrucciones de descarga y conversión del dataset.

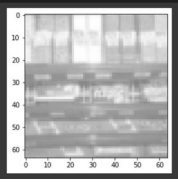
```
[ ]  1 # function to show an image
     2 def imshow(img):
     3     img = img / 2 + 0.5  # unnormalize
     4     npimg = img.numpy()
     5     plt.imshow(np.transpose(npimg, (1, 2, 0)))
     6
     7 import matplotlib.pyplot as plt
     8 import numpy as np
     9 import torchvision
    10
    11 # get some random training images
    12 dataiter = iter(trainloader)
    13 images, labels = dataiter.next()
    14
    15 #concatenate images
    16 imgconcat = torchvision.utils.make_grid(images)
    17
    18 # show images
    19 imshow(imgconcat)
    20 # print labels
    21 print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```

```
TallBuilding Coast Kitchen Industrial Industrial Highway
```

Por último, se añade una parte bonus, donde se explica la transformación de los datos, explicando la librería transform de PyTorch. Las trasnformaciones entre otras serán:

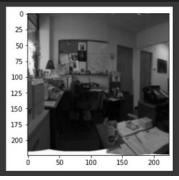
- Resize:

```
[ ]  1 #resizing transform
     2 import torchvision.transforms as transforms
     3
     4 Resizing_factor = (64,64)
     5 transform = transforms.Compose([transforms.Resize(Resizing_factor), transforms.ToTensor()])
     6
     7 to_pil_image = transforms.ToPILImage()
     8
     9 img = to_pil_image(images[0])
    10 imtran = transform(img)
    11 imshow(imtran)
```

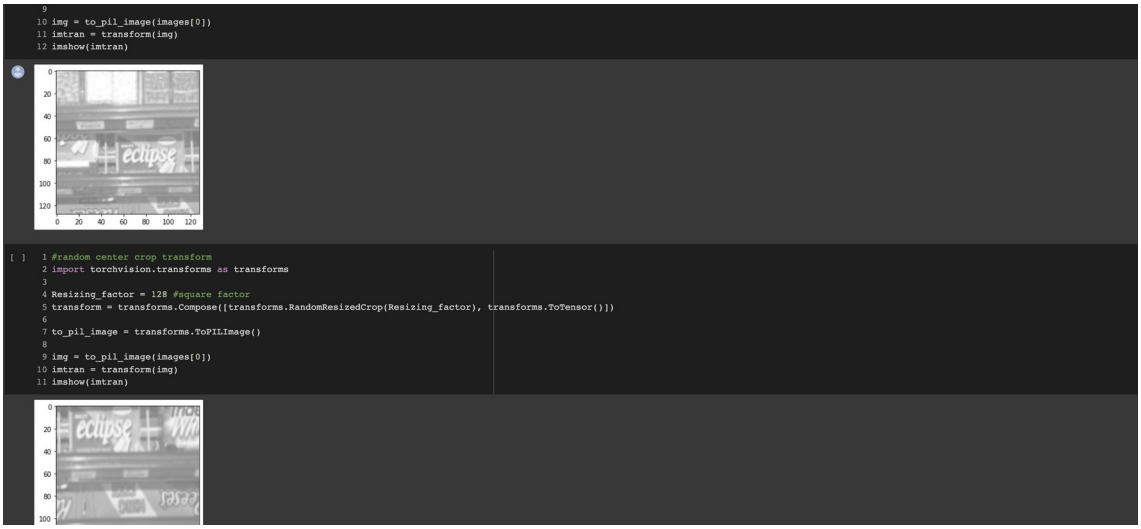


- Normalización:

```
[ ]  1 #normalizing transform
     2 import torchvision.transforms as transforms
     3
     4 transform = transforms.Compose(
     5     [transforms.ToTensor(),
     6      transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))])
     7
     8 to_pil_image = transforms.ToPILImage()
     9
    10 img = to_pil_image(images[0])
    11 imtran = transform(img)
    12 imshow(imtran)
```



- Crop(s):



Posteriormente, se explica como se pueden aplicar distintas transformaciones sucesivas a los datos. En este detalle, explica buenas prácticas a la hora de generar datos (imágenes) como hacer que todas tengan las mismas dimenesiones (cosas que se pueden alcanzar con un resize o con un crop) y normalizarlas para un entrenamiento óptimo.

Ejercicio 3 (Notebook: 3. network_p37.ipynb)
En este ejercicio se estudiará la topología de una red neuronal convolucional (CNN), de la mano de PyTorch como framework para poder crearla, entrenarla y obtener resultados de ella. Se explican los pasos y la definición de una CNN:

- Definir la estructura de la red, que tendrá algunos parámetros entrenables (i.e. weights*).
- Definir la secuencia de procesado de los datos para obtener una salida (i.e. *forward pass*).
- Calcular la función de pérdidas que indique la precisión de la salida con respecto a nuestras anotaciones (i.e. *loss function*).
- Calcular los gradientes de retropropagación ejecutando en modo inverso la red (i.e. *backward propagation*).
- Actualización de los parámetros de la red acorde a los gradientes anteriores.

Posteriormente, en este notebook se creará una red neuronal convolucional. Como tal, lo primero que se deberá de saber es que se ha definido una clase “Net” la cual actúa como “wrapper” de la funcionalidad que alberga PyTorch para poder crear redes neuronales. Este wrapper estará basado en el paquete **torch.nn**.

El paquete **torch.nn** tiene una serie de métodos para definir las capas de la red neuronal, los cuales son explicados en el notebook.

En primer lugar, tenemos la clase que define las capas convolucionales 2D. Esto se hace mediante la función Conv2d, que tiene como argumentos principales:

- **in_channels**: valor para la tercera dimension del tensor de entrada (e.g. 3 para imágenes RGB, 1 para imágenes en Gris).
- **out_channels**: número de mapas de salida (i.e. número de convoluciones o *kernels* que aplicamos sobre los datos de entrada).
- **kernel_size**: tamaño del *kernel* aplicado (tamaño x tamaño)
- **stride:** desplazamiento de la aplicación del operador de convolución
- **padding**: tipo de padding aplicado

Posteriormente explica otros tipos de redes, como las Linear, las cuales cabe destacar su propiedad fully connected, que hace que todas las neuronas estén conectadas. Esto puede afectar a la convolución y dar pie a redes que produzcan overfitting.

También se tienen etapas, como la explicada en el notebook, MaxPooling2D, que hace una reducción en la dimensionalidad de los datos, y que tiene los siguientes argumentos de interés:

- **kernel_size**: tamaño del *kernel* aplicado (tamaño x tamaño).
- **stride**: desplazamiento de la aplicación del operador de convolución.
- **padding**: tipo de padding aplicado.

Posteriormente se realiza una definición de la red mediante la clase “Net”, el wrapper que se comentó al principio de esta sección.

Una vez definida la red, se explica la primera parte del funcionamiento de la red. En este caso, la idea es sencilla, se definen una serie de valores aleatorios con dimensión igual a la entrada de la red. Estos valores se definirán como los valores de entrada de la red. La red producirá una salida, la cual podremos ver como un vector de valores, el cual, dependiendo del problema, se habrá de interpretar.

Para que la red aprenda, tal y como pasa con las especies de la naturaleza, se debe “recompensar” o “castigar” las acciones que esta desempeñe. Si las acciones son beneficiosas (una correcta clasificación de una imagen) se ha de notificar a la red que va por el camino correcto. Para eso se tienen funciones (de pérdida) que le dicen al algoritmo como de bien (o de mal) va encaminado.

Función de pérdidas (loss function)

Una función de pérdida toma el par de entradas (salida, objetivo) y calcula un valor que calcula qué tan lejos está la salida del objetivo. Existen varias funciones de pérdida en el paquete *nn*.

Una pérdida simple es: `nn.MSELoss` que calcula el error cuadrático medio entre la entrada y el objetivo. Una pérdida simple es: `nn.CrossEntropyLoss` que calcula el error entropía cruzada entre la entrada y el objetivo.

A continuación se muestra un ejemplo de ejecución

```
#generate fake input/output
output = net(input)
target = Variable(torch.arange(1, 11))  # a dummy target, for example
target = target.to(torch.float32)

#define criterion for loss function
criterion = nn.MSELoss()

#apply loss function
loss = criterion(output, target)
print(loss)
```

✓ 0.1s Python

```
tensor(38.3632, grad_fn=<MseLossBackward0>)
```

Una vez que hemos “premiado” o “castigado” a la red, los pesos de la misma (parámetros que la red “aprende”) deberán de ser actualizados. Para eso se usa el algoritmo de backpropagation:

Retropropagación (backpropagation)

Para poder calcular el error cometido por la red en cada etapa, debemos ejecutar la red en modo inverso mediante la funcion `loss.backward()`. Esta función se calcular a partir de la función `forward` que hemos proporcionado en la definición de la red.

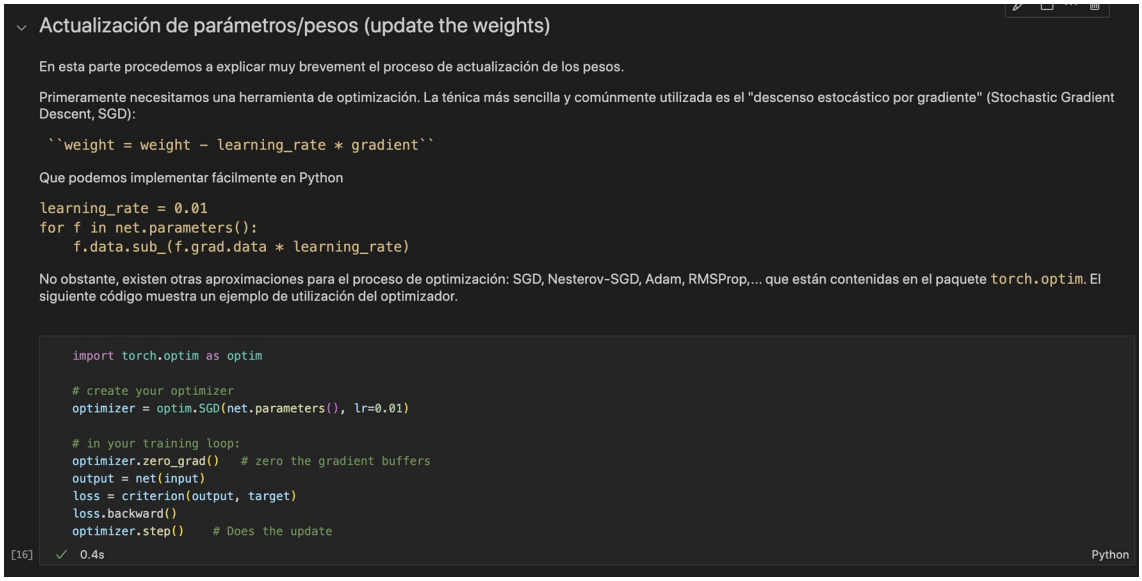
Para retropropagar el error, han de seguirse los siguientes pasos:

```
# You need to clear the existing gradients though, else gradients will be accumulated to existing gradients
net.zero_grad()     # zeroes the gradient buffers of all parameters

#have a look at conv1's bias gradients before and after the backward.
print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

✓ 0.4s Python

```
conv1.bias.grad before backward
tensor([0., 0., 0., 0., 0., 0.])
conv1.bias.grad after backward
tensor([ 0.0184,  0.0432,  0.0604, -0.0113, -0.1083,  0.0823])
```

El cual computará el descenso por gradiente, dando a cada neurona un valor al que ajustarse.
Una vez calculado el valor de ajuste, se tendrá en cuenta la proporción de aprendizaje de la red. Este valor dice como de rápido bajamos por el descenso de gradiente, es decir, la velocidad con la que tratamos de acercarnos al mínimo local mas cercano que obtenemos de la función de activación:



Actualización de parámetros/pesos (update the weights)

En esta parte procedemos a explicar muy brevement el proceso de actualización de los pesos.

Primeramente necesitamos una herramienta de optimización. La ténica más sencilla y comúnmente utilizada es el "descenso estocástico por gradiente" (Stochastic Gradient Descent, SGD):

``weight = weight - learning_rate * gradient``

Que podemos implementar fácilmente en Python

```
learning_rate = 0.01
for f in net.parameters():
    f.data.sub_(f.grad.data * learning_rate)
```

No obstante, existen otras aproximaciones para el proceso de optimización: SGD, Nesterov-SGD, Adam, RMSProp,... que están contenidas en el paquete `torch.optim`. El siguiente código muestra un ejemplo de utilización del optimizador.

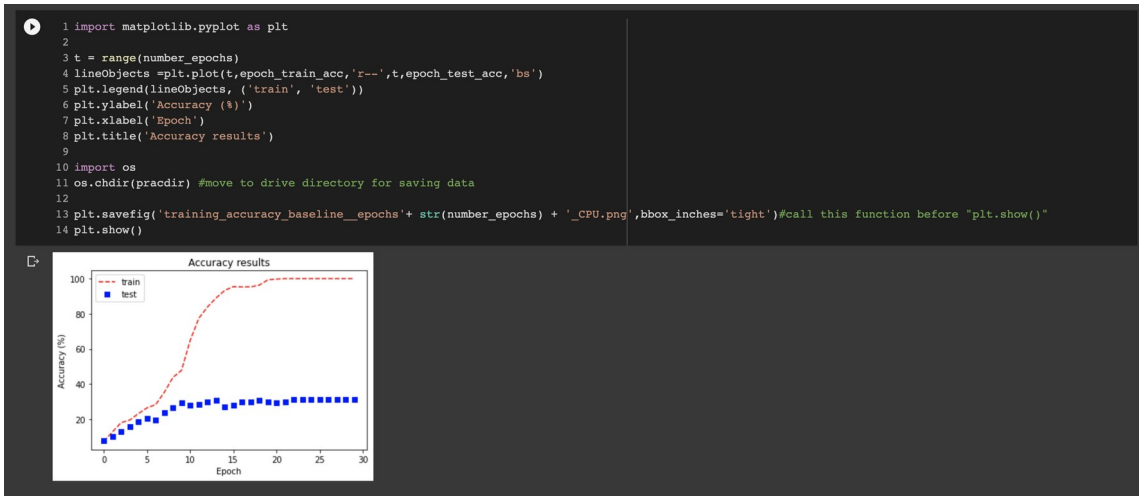
```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

[16] ✓ 0.4s Python

Y con eso, se ajusta, para una época, los pesos de una red neuronal.

Ejercicio 4

Tal y como se indica en la transparencia 21 de la práctica, los resultados que se tienen ejecutando el cuaderno **4.training_baseline_p37_cpu.ipynb** tienen una precisión de entrenamiento del 100%, pero un 32 ~ 35% de validación/test. Esto nos hace pensar que la red no generaliza correctamente cuando datos nuevos entran en la entrada de la red.



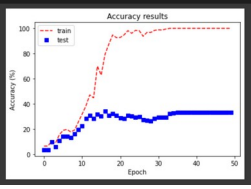
Para ello se proponen unas sugerencias a modo de mejora:

1. **Estudio del efecto del parámetro batchsize en 50 épocas.**

   Batch_size = 2

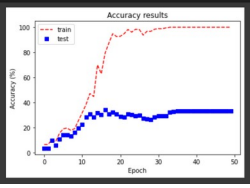
Batch_size = 4

```
import matplotlib.pyplot as plt

t = range(number_epochs)
lineObjects =plt.plot(t,epoch_train_acc,'r--',t,epoch_test_acc,'bs')
plt.legend(lineObjects, ('train', 'test'))
plt.ylabel('Accuracy (%)')
plt.xlabel('Epoch')
plt.title('Accuracy results')

import os
os.chdir(pracdir) #move to drive directory for saving data

plt.savefig('training_accuracy_baseline__epochs'+ str(number_epochs) + '_CPU.png',bbox_inches='tight')#call this function before "plt.show()"
plt.show()
```



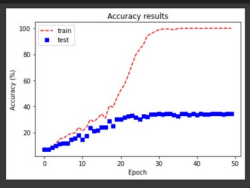
Batch_size = 8:

```
import matplotlib.pyplot as plt

t = range(number_epochs)
lineObjects =plt.plot(t,epoch_train_acc,'r--',t,epoch_test_acc,'bs')
plt.legend(lineObjects, ('train', 'test'))
plt.ylabel('Accuracy (%)')
plt.xlabel('Epoch')
plt.title('Accuracy results')

import os
os.chdir(pracdir) #move to drive directory for saving data

plt.savefig('training_accuracy_baseline__epochs'+ str(number_epochs) + '_CPU.png',bbox_inches='tight')#call this function before "plt.show()"
plt.show()
```



Batch_size = 16:

```
import matplotlib.pyplot as plt

t = range(number_epochs)
lineObjects =plt.plot(t,epoch_train_acc,'r--',t,epoch_test_acc,'bs')
plt.legend(lineObjects, ('train', 'test'))
plt.ylabel('Accuracy (%)')
plt.xlabel('Epoch')
plt.title('Accuracy results')

import os
os.chdir(pracdir) #move to drive directory for saving data

plt.savefig('training_accuracy_baseline__epochs'+ str(number_epochs) + '_CPU.png',bbox_inches='tight')#call this function before "plt.show()"
plt.show()
```

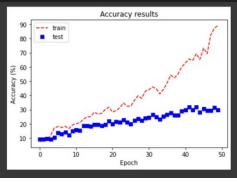


Batch_size = 32:

Análisis de precisión y función de pérdidas

A continuación pintamos las variables que hemos acumulado durante el entrenamiento.

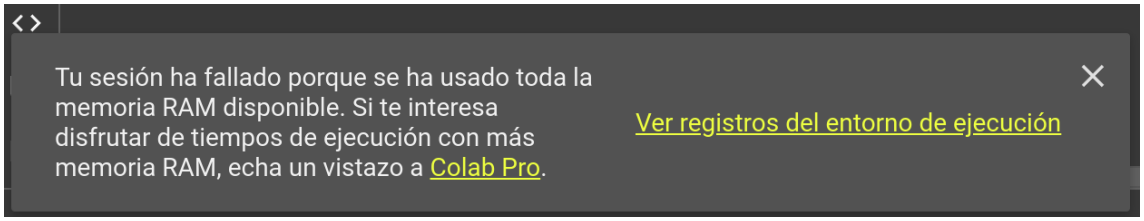
```
import matplotlib.pyplot as plt

t = range(number_epochs)
lineObjects =plt.plot(t,epoch_train_acc,'r--',t,epoch_test_acc,'bs')
plt.legend(lineObjects, ('train', 'test'))
plt.ylabel('Accuracy (%)')
plt.xlabel('Epoch')
plt.title('Accuracy results')

import os
os.chdir(pracdir) #move to drive directory for saving data

plt.savefig('training_accuracy_baseline__epochs'+ str(number_epochs) + '_CPU.png',bbox_inches='tight')#call this function before "plt.show()"
plt.show()
```



Vemos que ninguno de los valores da una ventaja diferencial a la hora de cambiar dicho parámetro. De hecho, el valor que mejores resultados ha dado ha sido el dado por defecto, en este caso **batch_size = 4**.

2 Estudio del efecto del parámetro Resizing_factor.

Utilizamos un intervalo de valores para este parámetro entre 32 y 256, para las ejecuciones de 32, 64 y 128 (parámetro por defecto) no tuvimos problema, pero para la de 256 constantemente nos aparece este error. No he sido capaz de resolverlo.



En cuanto al resto de valores, no hemos conseguido obtener ningún valor que particularmente de unos procentajes de accuracy mejorados. Hemos dejado el valor 128 como el mejor, aunque se tienen todos los resultados en los notebooks.