Indexación, búsqueda y análisis en repositorios multimedia: imagen-video

Juan Carlos San Miguel

Trabajo práctico: indexación de imágenes con CNNs

- Objetivos
- Entregas
- Herramientas: Google Colaboraroty/ Pytorch
- Parte1: Desarrollo
  - Entorno de trabajo
  - Manejo de datasets
  - Definición de redes neuronales convolucionales
  - Entrenamiento sin/con GPUs
- Parte 2: Mejoras
- Anexo: errores comunes

Trabajo práctico: Objetivos

- Iniciarse en el desarrollo de redes neuronales con PyTorch.
- Conocer los comandos y terminología asociados a las distintas partes para la generación de una red neuronal convolucional.
- Propuesta de mejoras de sobre una red inicial, aplicando los conceptos aprendidos en la práctica e investigando en internet.

Trabajo práctico: entregas

- Se realizan por parejas (se debe indicar los nombres en el PDF)
- La práctica consta de dos partes
  - Parte 1 (obligatoria, 4 puntos): realizar los tutoriales de la parte 1. Deberá entregar los **ficheros de código IPython (*.ipynb)** donde se visualice el resultado de las ejecuciones.
  - Parte (opcional, 6 puntos): se propondrán una serie de actividades que extienden los resultados de los scripts del apartado anterior. Deberá entregar los **ficheros de código IPython (*.ipynb**) más una descripción de las mejoras en una **memoria con un único fichero PDF.**

Trabajo práctico: Herramientas

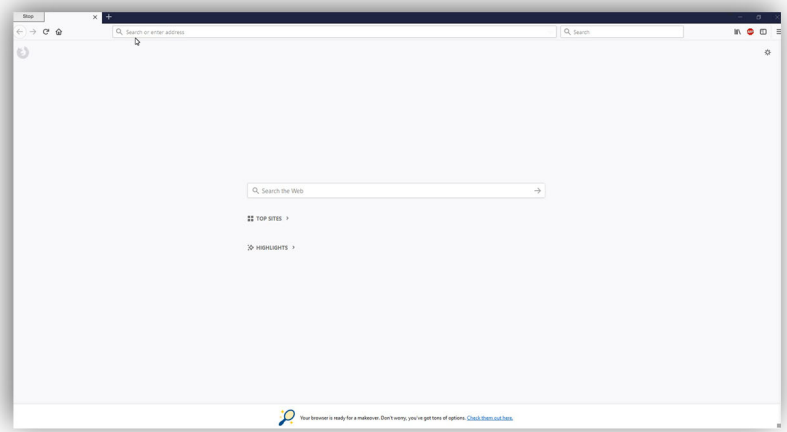
- **Pytorch** (http://pytorch.org/) librería de código libre basada en Python Fácil uso y prototipado de algoritmos de Machine Learning y redes neuronales. Desarrollada principalmente por Facebook (grupo de IA)
- **Jupyter Notebook** (http://jupyter.org/) software que permite ejecutar de manera interactiva código de python (i.e. visualizando el output de bloques de código en tiempo de ejecución)
- **Google Colaboratory** (https://colab.research.google.com) es un entorno de trabajo en la nube que proporciona recursos de computación gratuitos


Trabajo práctico: Google Colaboratory

- Google Colaboratory proporciona una máquina virtual con 2 CPUs Intel 2.2GHz, 12GB RAM, disco duro de ~100GB y una GPU Tesla T4
  - Estos recursos se otorgan de manera continuada durante aprox. 12 horas (a partir de ese tiempo, se desasigna la máquina virtual y se reasigna otra similar si se vuelve a solicitar)
  - Esta limitación no es un problema para esta práctica, puesto que las ejecuciones necesarias requieren 3-4 horas como máximo.
- Requisitos:
  - Una cuenta @gmail para acceder a los servicios de Google
    Se recomienda crearse una cuenta específica para la asignatura para reducir el riesgo de pérdida o acceso malintencionado a nuestros datos personales.
  - Un navegador con acceso a internet para conectarnos a la máquina virtual de Google Colaboratory (e.g., Google Chrome y Mozilla Firefox)
- Dudas comunes: https://research.google.com/colaboratory/faq.html

Trabajo práctico: Google Colaboratory

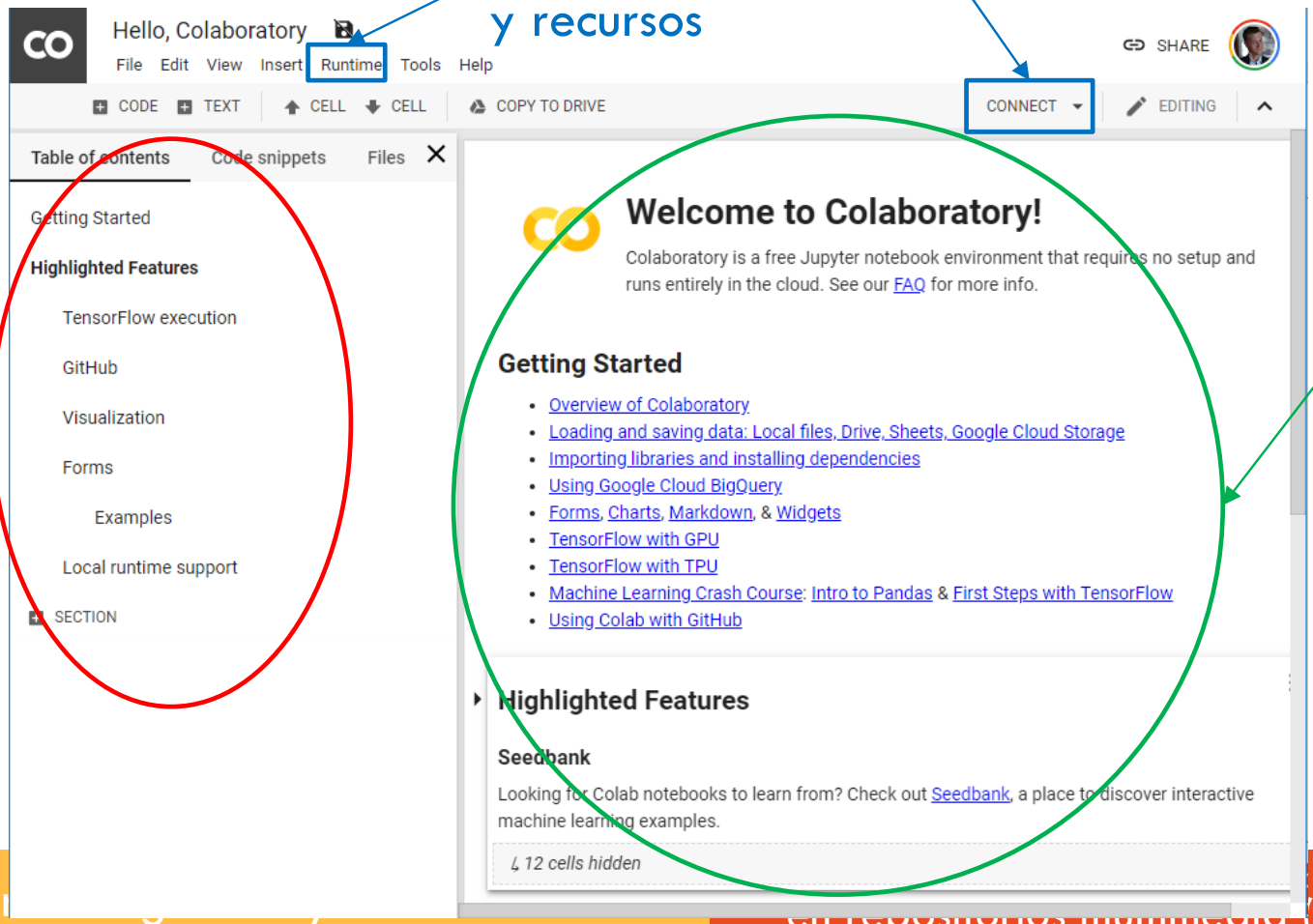
- Inicio:
  - Abrir el siguiente enlace para iniciar Google Colab con un ejemplo https://colab.research.google.com/notebooks/welcome.ipynb



Video con los pasos disponible en http://recordit.co/1lzUat4o36

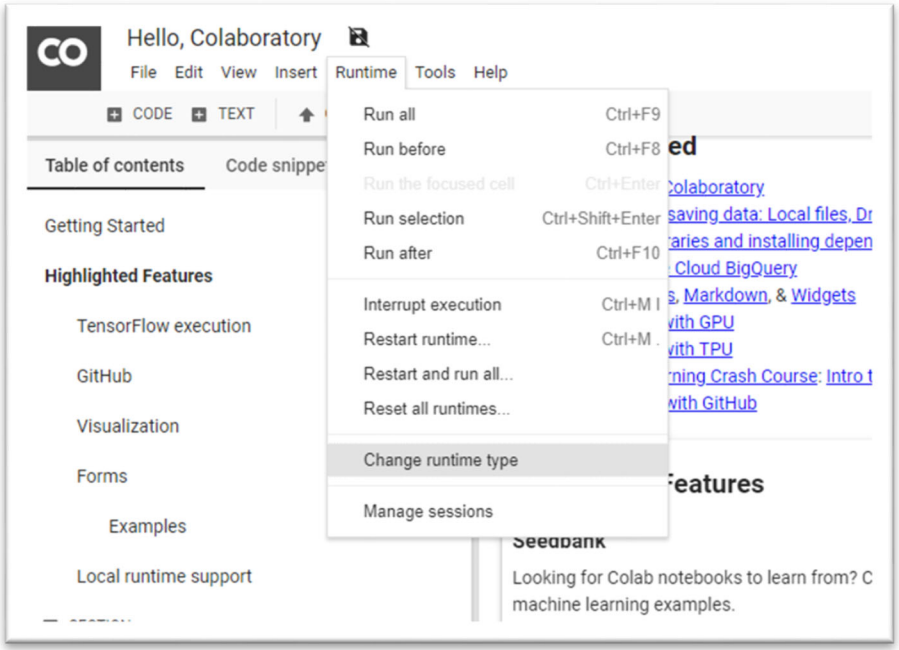
Trabajo práctico: Google Colaboratory

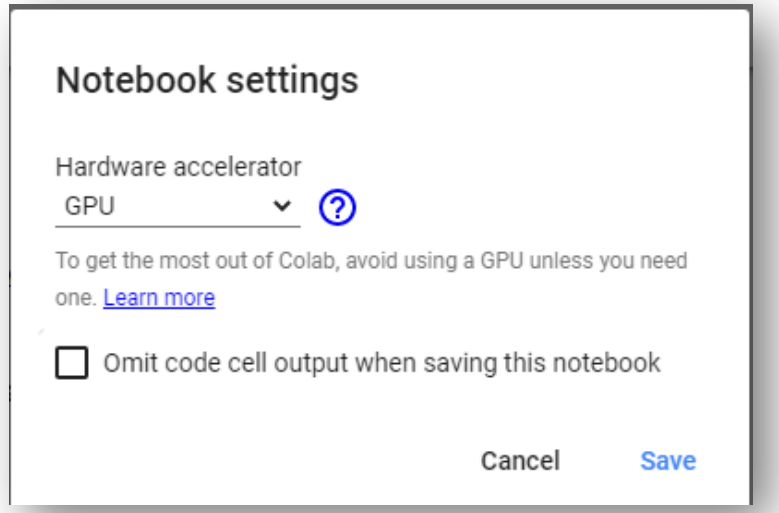
- Inicio: interfaz



Trabajo práctico: Google Colaboratory

- Inicio: seleccionar recursos para la máquina virtual

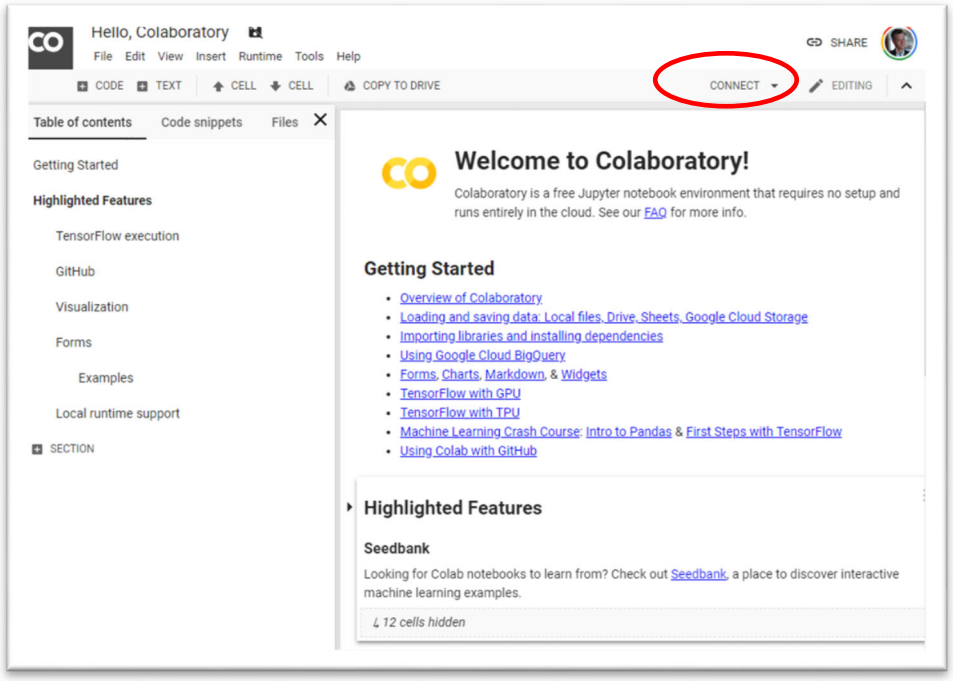


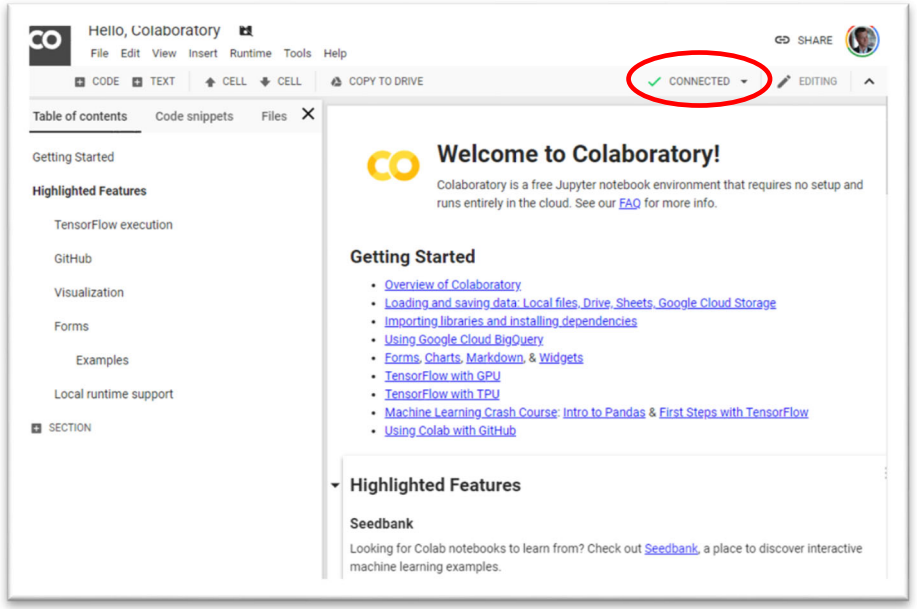


Trabajo práctico: Google Colaboratory

- Inicio: conectar la máquina virtual con los recursos seleccionados

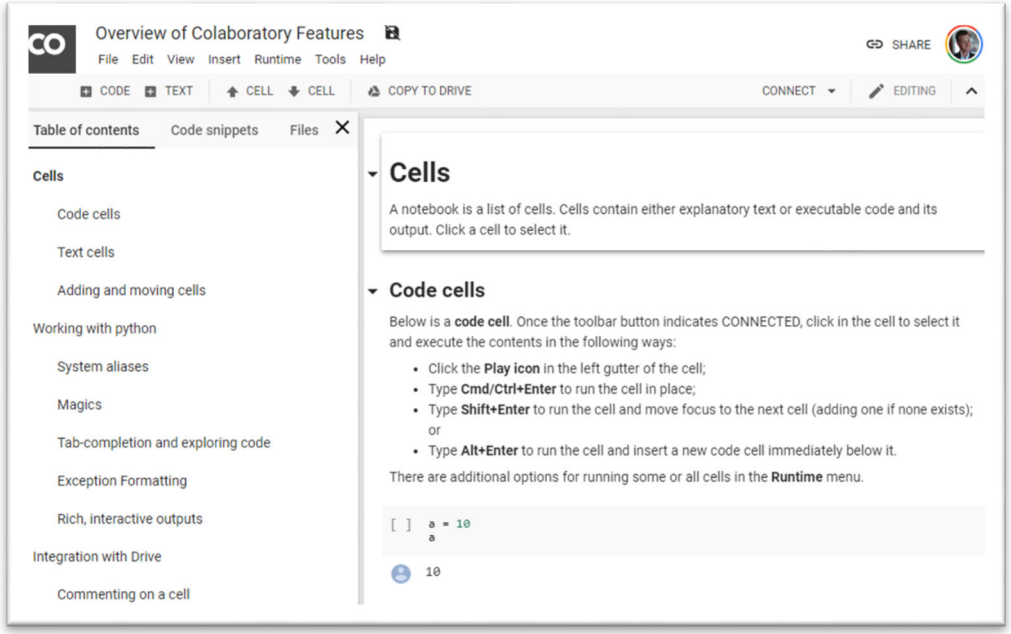
Hacer click en “CONNECT” y verificar que estamos conectados (tick verde)





Trabajo práctico: Google Colaboratory

- Tarea: realizar el tutorial de uso básico de Google Colab
- Tiempo: 15 minutos



https://colab.research.google.com/notebooks/basic_features_overview.ipynb

Trabajo práctico: Pytorch

- Librería código libre para Python
- Algoritmos de Machine Learning y redes neuronales
- Reutilización de paquetes Python tales como NumPy, SciPy, OpenCV,…
- Fácil adaptación de código para uso de GPUs
- Tutoriales
  - Python https://www.python.org/about/gettingstarted/
  - Pytorch http://pytorch.org/tutorials/

- Esta práctica se ha realizado con Python versión 3.7, Pytorch 1.10 y CUDA 11 (no se garantiza la ejecución con otras versiones)

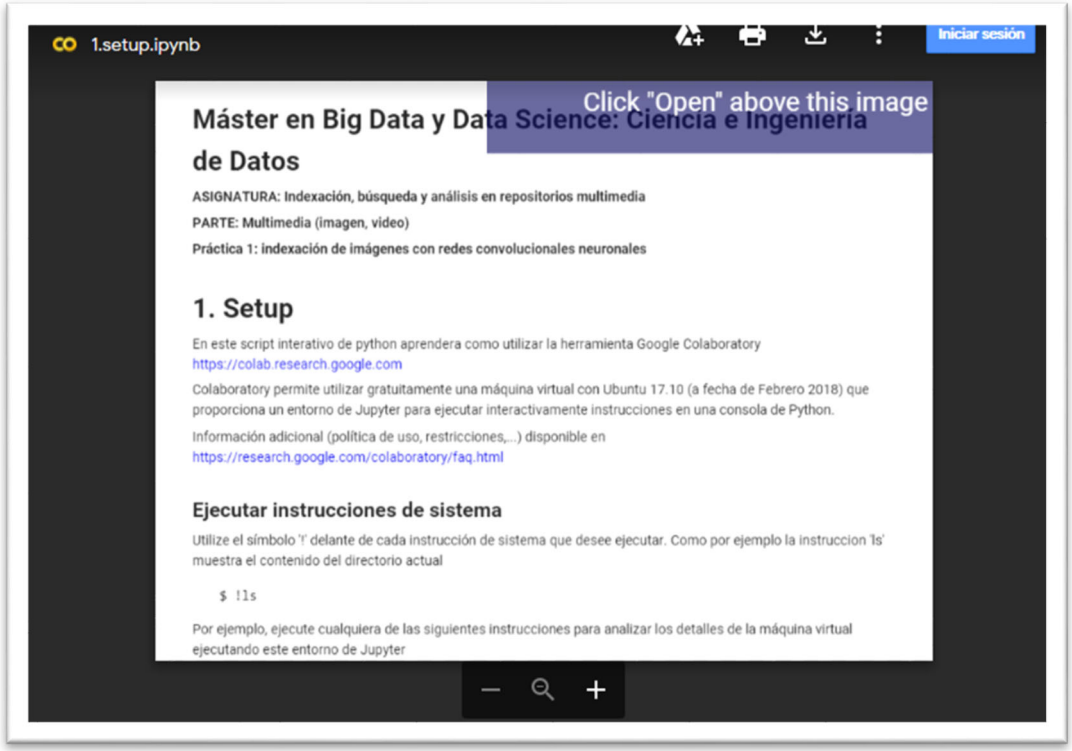
Trabajo práctico: Parte1- Desarrollo

- Establecer entorno de trabajo
  - Establecer el entorno de trabajo que permita utilizar Pytorch de manera fluida y guardar los datos generados en su unidad de Google Drive.
  - En concreto se describen los siguientes pasos:
    - Ejecutar instrucciones de sistema y verificar la existencia de GPU
    - Instalación de paquetes necesarios (Pytorch, Pillow,…)
    - Conectar la máquina virtual con su unidad Google Drive
    - Operaciones más comunes

Abra el siguiente script (https://bit.ly/3Gp28d5)
https://drive.google.com/open?id=1p4bAZGCe_t4NJNZXo1J77Cj83JA8m16F

Trabajo práctico: Parte1- Desarrollo

- Establecer entorno de trabajo
  - Este script está protegido contra escritura
  - Para poder modificarlo, deberá generarse una copia local en su Google Drive. Para ello:
    - Logearse en Google Drive
    - Abra su fichero en el "playground" (una vez abierto, seleccione "Open in playground")
    - Copie a su unidad con la opción "Copy to Drive" en el menú superior.



Video con los pasos http://recordit.co/1IzUat4o36

Trabajo práctico: Parte1- Desarrollo

- Manejo de datasets
  - Preparación del entorno de trabajo (resumen de lo anterior)
  - Datasets por defecto en PyTorch: descarga, conversión a tensores y visualización.
  - Datasets genéricos: descarga, conversión a tensores y visualización
  - Bonus: transformaciones de los datos

Abra el siguiente script (https://bit.ly/32WYUz8)
https://drive.google.com/open?id=12LsMfZGsucpDmFTeGbnW9rc4n1GZ-Ib6

Trabajo práctico: Parte1- Desarrollo

- Definición de redes neuronales convolucionales
  - Preparación del entorno de trabajo
  - Diseño de capas: convolucional, activación ReLU, *pooling* y *fully-connected*
  - Ejecución de una red (*forward pass*).
  - Definición de una función de pérdidas (loss function)
  - Retropropagación para entrenamiento (*backpropagation*)
  - Inicialización y actualización de parámetros/pesos (*weight initialization/update*)

Abra el siguiente script (https://bit.ly/3IHCsR0)
https://drive.google.com/open?id=1kOrdpI27NWA9AaXp4J2Rn0wymmkL7sm-

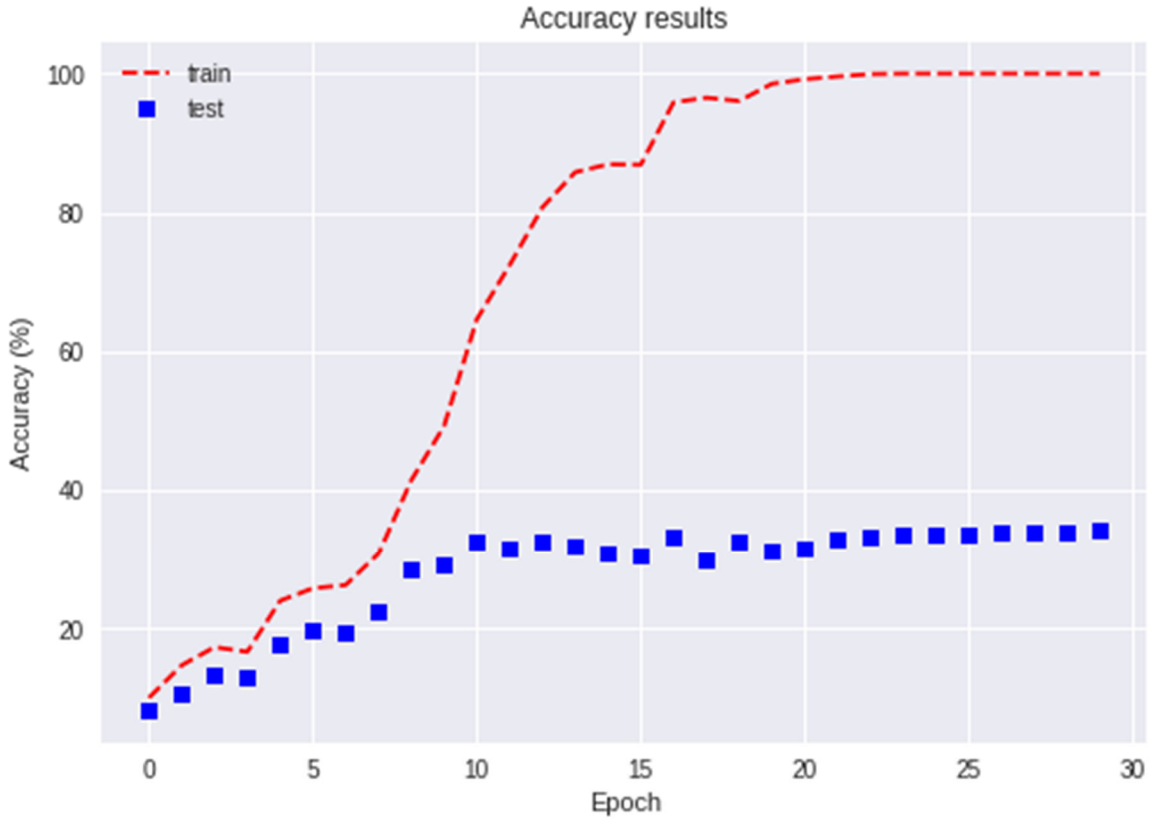
Trabajo práctico: Parte1- Desarrollo

- Entrenamiento sin GPUs (versión CPU)
  - Preparación del entorno de trabajo (resumen minimalista de la parte 1)
  - Descarga y preparación del dataset (resumen minimalista de la parte 2)
  - Definición de la red (resumen de la parte 3). Aquí utilizamos el fichero *mylenet.py* para cargar la red. Definición de una función de coste (*loss function*) y optimizador
  - Proceso iterativo de entrenamiento mediante *épocas*
  - Muestra de resultados tras finalizar el entrenamiento

Abra el siguiente script (https://bit.ly/3orrHEl)
https://colab.research.google.com/drive/11N_V1KL_elpRR4p5zO8bqupYE2Alo0I9

Trabajo práctico: Parte1- Desarrollo

- Entrenamiento sin GPUs (versión CPU)
  - Tras ejecutar el script con un *batchsize=4*, en aprox. ~16 minutos se obtiene una precisión de 100% train y 32-35% test, tras 30 iteraciones/épocas.



Trabajo práctico: Parte1- Desarrollo

- Entrenamiento con GPUs
  - Similar al entrenamiento con CPUs
  - Una gran ventaja de PyTorch es su sencillo soporte para cálculo intensivo sobre GPUs mediante la librería CUDA. Para utilizar GPUs en operaciones con tensores, se requiere modificar el código añadiendo ".cuda" a la operación correspondiente. Se muestra en el siguiente ejemplo:

```
>> net = Net() #creación de la red para ejecutar en CPU
>> net.cuda()  #indicamos que debe ejecutarse en modo GPU
```

Abra el siguiente script (https://bit.ly/3y1kpKE)
https://drive.google.com/open?id=1HsCUBa3Cs8fXsDHZe1yINLCFoSFLz-Xk

Trabajo práctico: Parte1- Desarrollo

- Entrenamiento con GPUs
  - Tras ejecutar el script GPU con un *batchsize=4*, en aproximadamente ~10 minutos deberá obtener unos resultados de precisión (100% en train y 32-35% en test) tras 30 iteraciones/épocas.
  - Observe que el tiempo de ejecución en GPU no es sustancialmente diferente al tiempo en CPU, contrariamente a lo que se debería esperar. Esto se debe a que la red descrita en *mylenet.py* no es suficientemente profunda ya que consta de pocas capas (dos convolucionales y tres fully-connected).
  - La ventaja del uso de GPUs se observa claramente en grandes redes donde se puede obtener una aceleración en un orden de magnitud 8-10x.

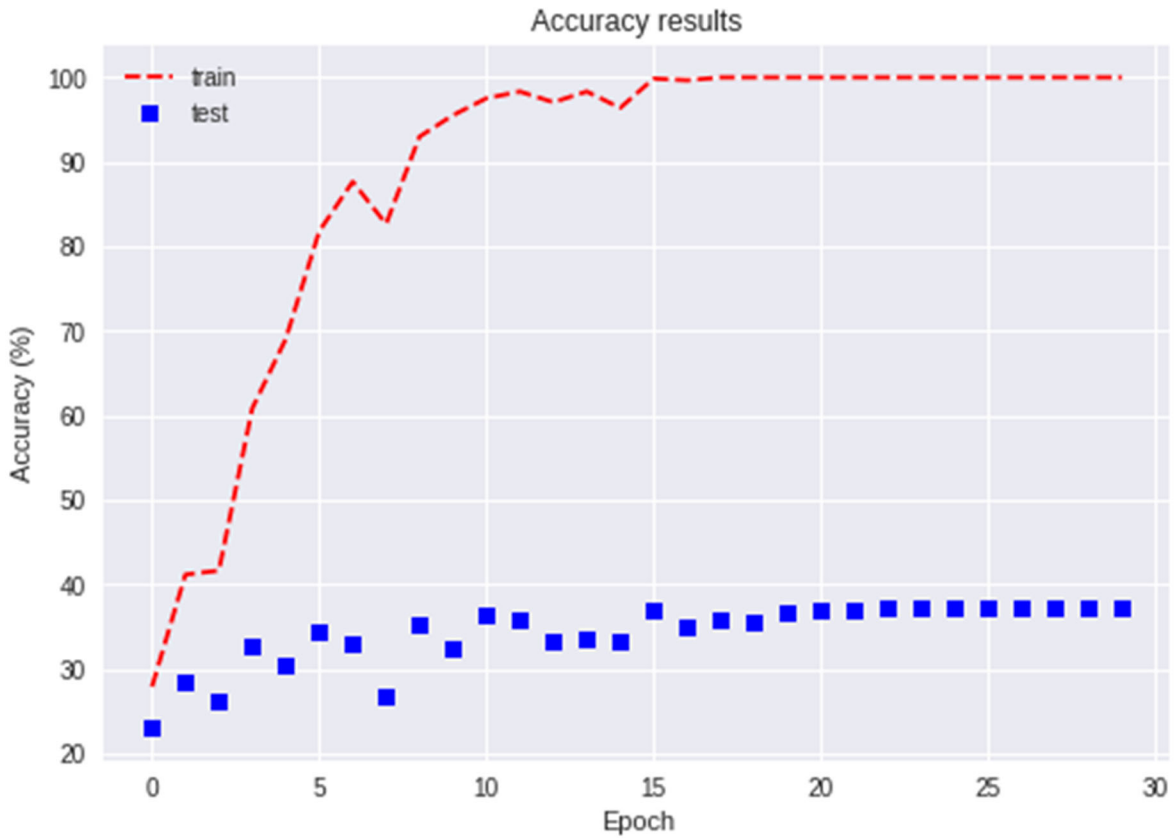
Trabajo práctico: Parte2- Mejoras

Objetivo: mejorar el rendimiento en los conjuntos train/test de la red proporcionada en mylenet.py sobre el dataset *Scene15*.

- Para esta parte no se proporciona scripts interactivos.
- Aplicar mejoras poniendo en práctica lo aprendido en clase
- **Entrega por cada mejora:**
  - Script ipynb con los resultados generados
  - Documentación con una descripción de la(s) propuesta(s) y su impacto (es decir, compare los resultados antes y después de añadir la mejora)

Trabajo práctico: Parte2- Mejoras

- Resultado obtenido en la parte 1
  - Punto de partida parte 2
  - Precision
    - ~100% train y ~35% test
  - Existe overfitting
  - Dataset train pequeño (1500 imag, 15 categorias)
  - Mucho margen de mejora…



Trabajo práctico: Parte2- Mejoras

- Sugerencias de mejora (1/2)
  - (hasta 1.5 puntos) Estudio del efecto del parámetro *batchsize* en 50 épocas
  - (hasta 1.5 puntos) Estudio del efecto del parámetro *Resizing_factor*
  - (hasta 2 puntos) Explore el uso de normalización de imágenes como última operación de una transformación compuesta. Ejemplo en el siguiente fichero: https://github.com/pytorch/examples/blob/master/imagenet/main.py
  - (hasta 2 puntos) Explore y compare distintas estrategias de optimización (más info en http://pytorch.org/docs/master/optim.html)
  - (hasta 2 puntos) Explore y compare una aplicación progresiva de un factor learning rate mediante el elemento *lr_scheduler* (http://pytorch.org/docs/master/_modules/torch/optim/lr_scheduler.html)

Trabajo práctico: Parte2- Mejoras

- Sugerencias de mejora (2/2)
  - (hasta 3 puntos) Aplique técnicas de *Data Augmentation* para mejorar la calidad de los datos de entrenamiento. Para realizar nuevas transformaciones, se sugiere utilizar la funcionalidad Compose descrita en http://pytorch.org/docs/master/torchvision/transforms.html
  - (hasta 3 puntos) Aplique *fine-tuning* al problema para adaptar una red 'relativamente' compleja al problemade esta práctica. Más información en https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
  - (hasta 3 puntos) Modificar la arquitectura original de *mylenet.py* añadiendo un número mayor de capas convoluciones con un mayor campo de visión (i.e. mayor tamaño los filtros convolucionales)

Anexo: errores comunes

- **Listado de Errores comunes en:**

  http://www-vpu.eps.uam.es/~jcs/bigdata/FAQ_errores_comunes.pdf