Desafío: Clasificación de símbolos musicales con redes neuronales convolucionales

DLSI

1. Introducción

1. Como aplicación práctica usaremos el concepto de OCR aplicado a notación musical **símbolos aislados (OMR)**.

2. Nuestro **modelo** será **mejor** cuanto **mayor tasa de aciertos** obtenga a partir de los mismos datos que otros modelos.

3. Se **presentará un breve informe** individual de los **pasos** realizados y los **resultados** obtenidos. Se usará el sistema de documentos de Google (GMail) para ello debes hacer clic en este enlace y facilitar tu dirección.

4. Para la implementación de la práctica se usará el lenguaje `Python` usando básicamente la biblioteca de `Deep Learning`, `Keras` (documentación http://keras.io/, breve video-tutorial). Se proporcionará un ejemplo inicial básico para mejorarlo hasta la fecha de entrega de esta práctica.

2. Desafío

- La base de datos que se usará consiste en la descripción de las símbolos musicales mediante imágenes aisladas obtenidas en una tablet (enlace al artículo Recognition of Online Handwritten Music Symbols) que está disponible en el Campus Virtual. Los datos están divididos en 32 directorios con muestras de una sola clase cada uno.

- Debemos **probar** el ejemplo básico de la red convolucional, y renombrarlo a `homus_cnn.py`, como punto de partida para mejorar su tasa de aciertos ajustando alguno de sus parámetros como el número de capas, neuronas por capa, o bien, cambiando la arquitectura general de la red (ver cuadro de Evaluación 3).

- Se realizará una **corrección presencial** y se verificará el resultado final con una nueva batería de imágenes (test).

3. Evaluación

Se valorará que el **informe** sea resumido (máximo 4 páginas), claro, completo y con actividad durante el cuatrimestre; además de los ajustes y mejoras realizadas (ver cuadro).

	Valoración	Descripción
Básico	hasta 5 puntos	▪ Funcionamiento correcto con la red entregada (`homus_cnn.h5`). ▪ Aplicación de la validación cruzada 10-CV. ▪ Ajustes mínimos en los parámetros de la red inicial. ▪ Algún estudio comparativo con test estadísticos. ▪ Informe correcto y con actividad durante el cuatrimestre.
Medio	hasta 3 puntos	Cambios menores respecto de la red inicial: ▪ Ajustes combinados (reescalado inicial de la imagen, número de capas y parámetros tanto de convolución como del resto de la red). ▪ Uso de técnicas de aumentado de datos. ▪ Ajuste de algoritmo de optimización (mejores resultados con menos épocas). ▪ Uso de la notación funcional de Keras (functional API) en toda la práctica. ▪ Comparativas entre los diferentes ajustes con verificación de resultados (test estadísticos).
Avanzado	hasta 2 puntos	Cambios significativos respecto de la red inicial: ▪ Interface web sencillo para probar el modelo (Ej.: vídeo - How to Deploy Keras Models to Production) ▪ Cambio o mejora significativa sobre la arquitectura de la red inicial (Ej. `SqueezeNet` o `ResNet`). ▪ Uso de redes `InfoGAN` o `AC-GAN` (`GAN - Generative Adversarial Networks`) para generar nuevos ejemplos o mejorar la clasificación. ▪ Comparación con otros clasificadores como ej. `Random Forest` ó `SVM` (paquete `sklearn`). ▪ Estudio de los cambios propuestos en caso de presencia de ruido (0 %, 20 %, 40 %). ▪ Aportaciones propias en alguna de las etapas de la construcción de la red o de la experimentación.

4. Entrega

Se comprimirán los dos ficheros siguientes en un TGZ que se entregará en el sistema de entregas de prácticas del DLSI en las fechas establecidas en la asignatura:

- `homus_cnn.py`: Contendrá el código completo de la práctica donde se destacará el tamaño de reescalado;
- `homus_cnn.h5`: Este archivo contendrá la mejor red neuronal propia programada para esta práctica y guardada con `model.save("homus_cnn.h5")`[1].

5. Instalación de la libraría Keras en sistemas GNU/Linux

A continuación se detallan los programas y librerías habituales para trabajar con el módulo de `Keras` en sistemas `GNU/Linux` basados gestores de paquetes `apt` (para otros sistemas las librerías necesarias son similares usando sus respectivos instaladores de programas).

Es recomendable usar un entorno virtual (ej.: `virtualenv` o `venv`) para mantener una instalación nueva y actualizada que no afecte al funcionamiento del sistema operativo. Si es posible instalar una versión reciente de Python (ej. 3.6.x).

5.1. Cómo instalar una versión reciente de Python

Los pasos a seguir en un sistema Linux con gestor de paquetes `.deb` es:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
sudo apt update
sudo apt install python3.6 python3.6-dev python-venv
```

5.2. Cómo crear y activar un entorno virtual

Vamos a crear un entorno virtual llamado `py36` y activarlo.

```
python3.6 -m venv py36
source ./py36/bin/activate
```

5.3. Cómo instalar los paquetes necesarios en nuestro entorno de Python

En este ejemplo usaremos el gestor `pip` pero también podemos usar el sistema Anaconda con su gestor `conda`. Si hemos activado el entorno virtual no necesitaremos usar `sudo` para la instalación de paquetes.

[1] Recordad que es importante que este fichero se corresponda con el tamaño de las imágenes de entrada predeterminado en `homus_cnn.py` para su corrección presencial

```
pip install pip -U
pip install keras
pip install h5py
pip install tensorflow
pip install Pillow
pip install matplotlib
pip install jupyter
pip install sklearn
```

También se puede instalar Tensorflow para el uso de la GPU de la tarjeta gráfica NVidia (CUDA, cuDNN) si se dispone de ella con `pip install tensorflow-gpu`