# University of Pisa

Symbolic Evolutionary Artificial Intelligence

**Porting of Tensorflow deep learning demos in JAX and performance comparison**

Students

Fabio Malloggi and Davide Vigna

Academic Year: 2023-2024

# Contents

**1 Introduction** **2**
- 1.1 Challenges and Project Goal . . . 3

**2 Frameworks** **4**
- 2.1 TF . . . 4
- 2.2 PyTorch . . . 5
- 2.3 Keras . . . 6
  - 2.3.1 Keras 3 . . . 6
- 2.4 JAX . . . 8
  - 2.4.1 Just in time compilation (JIT) . . . 8
  - 2.4.2 Automatic Differentiation (grad) . . . 9
  - 2.4.3 Automatic Vectorization (vmap) . . . 10
  - 2.4.4 Automatic Parallelization (pmap) . . . 10

**3 Environment Configurations** **12**
- 3.1 Requirements . . . 12
- 3.2 Colaboratory Cloud Environment . . . 14
- 3.3 Local Environment . . . 15
  - 3.3.1 System Requirements . . . 15
  - 3.3.2 Preliminary Configuration . . . 15
  - 3.3.3 Windows – JAX – GPU Incompatibility issue . . . 15
  - 3.3.4 WSL2 Local Configuration Wizard . . . 16
  - 3.3.5 WSL2 Local Environment Manual Configuration . . . 17

**4 Hands on** **20**
- 4.1 Basics - Python and Machine Learning . . . 21
- 4.2 Advanced - Deep Learning . . . 23
- 4.3 Custom comparison: CNN - LeNet5 on all frameworks . . . 25

**5 Results and performances** **27**
- 5.1 SIC_ML_Chapter_08_Coding_Exercises . . . 28
- 5.2 LeNet5_Frameworks_Comparison . . . 30

**6 References** **32**

## 1 Introduction

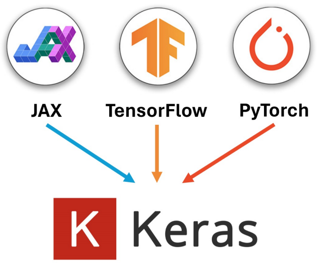
In recent years, the rapid growth of artificial intelligence has led various companies in the sector to develop specific frameworks for training deep neural networks. A Deep Learning framework offers building blocks for designing, training and validating deep neural networks through a high-level programming interface.

The diversity in neural network frameworks is mainly driven by:

- the need to address various application-specific requirements
- the presence of heterogeneous hardware, leading to the introduction of optimizations that significantly impact performance
- the different levels of user expertise allowing different level of usability and flexibility
- the iteroperability, modularity and integration of specific componenents and tools

In this project three of the main most used frameworks in the current state of the art are analyzed, comparing their methodologies and performances with respect to specific problems. Their names are TensorFlow, PyTorch and JAX. The focus will be particularly on JAX, a framework recently introduced by Google, to explore its main functionalities and to understand the key factors that make it more advanced and superior respect to TensorFlow.

In addition to these, a deep insight on Keras will be done. This famous Python library stands out for its simplicity and ease of use. Keras serves as an interface to several frameworks, such as the ones mentioned above, providing a high-level API that simplifies the process of building and training deep learning models. This allows the engineer to be focus better on the problem to be solved rather than on the techniques used to solve it. Keras is also beneficial because it allows for quick prototyping and experimentation due to its straightforward syntax and extensive documentation.



## 1.1 Challenges and Project Goal

This work aims to update and correct the material about the Machine Learning course from the Samsung Innovation Campus present in the GitHub repository available at the link:

`https://github.com/tn-220/SIC-Machine-Learning`

In the repository there are several Jupyter notebooks sorted by topic macro that use TensorFlow version 1.0 for training neural networks. The goal is to adapt these notebooks to the most recent version of TensorFlow, 2.16, correcting any operational errors. The intent is to recreate an updated, working version of the repository, using references to Keras whenever possible. Keras, acting as an interface between different deep learning frameworks, allows to analyze and switch in an easier way from a framework to an other. A brief report comparing the training performance of TensorFlow, PyTorch, and JAX across various hardware architectures and different kind of tasks will be presented at the conclusion of this work.

This project also has an important educational purpose: it wants to facilitate the learning of Computational Intelligence for those approaching this field for the first time, offering updated and well-functioning material for effective study. For this reason, some modifications have been applied to the original material, renaming and resorting some Jupyter Notebooks in a proper way following their topic argument. To facilitate the installation of the working environment a specific wizard is created to properly configure a WSL 2.0 in order to have complete access to hardware devices like GPU. This is explained extensively in chapter 3.

Additionally, custom materials have been provided, including specific implementations of neural networks using the three frameworks. These materials highlight the main differences in their implementations, with a particular emphasis on the characteristics of JAX.

# 2 Frameworks

## 2.1 TF

TensorFlow is one of the first open-source machine learning framework introduced in the modern era of AI, developed by the Google Brain team in 2015. It is designed to simplify the process of building, training, and deploying machine learning models, particularly neural networks thanks to its support for high-level APIs and the popular Keras library. It allows to perform complex numeric computations on large datasets using both CPUs and GPUs, providing a comprehensive set of APIs in multiple languages including Python and C, with ongoing development for Java, JavaScript, Julia, Matlab, and R. It also includes numerous pre-trained models and datasets such as MNIST, ImageNet, and COCO for immediate use. It allows the deployment of models on mobile and embedded devices. It also contains important tools like TensorBoard, a visualization toolkit for displaying images and model graphs, all of which make it a robust and versatile tool for developers and researchers in both academic and production environments. In recent years, TensorFlow has been updated to version 2.0 with the aim of simplifying its usage and meeting the most common needs of developers. This new version introduces several radical changes:

1. **Eager Execution** TensorFlow 1.x by default used a symbolic graph approach where a computation graph is defined and then executed within a session; Instead TensorFlow 2.x uses eager execution by default, which means operations are executed immediately. It is more intuitive and easy to debug.

2. **Keras Integration** The Keras API was not tightly integrated as the default API for model building in TensorFlow 1.x. The developers were encouraged to use other APIs or frameworks for defining and training models because of this lack. In version 2.x, Keras is the recommended way to define and train models because it provides a simple and intuitive interface for building neural networks.

3. **Automatic Differentiation** In the original version of TensorFlow, there was no possibility to customize training loops and advanced optimization algorithms. In TensorFlow 2.0, a more advanced and user-friendly automatic differentiation system has been introduced and thanks to the *tf.GradientTape* API is possible to customize the training procedure.

4. **Compatibility** TensorFlow 2.x maintains compatibility with TensorFlow 1.x models through the *tf.compat.v1* module, which allows users to migrate their existing TensorFlow 1.x code to TensorFlow 2.x gradually. However, it is not so easy to do the opposite operation (from 2.x to 1.x).

5. **API Simplification** In TensorFlow 2.x, numerous redundant and outdated APIs have been eliminated or merged, making the library more user-friendly and easier to learn.

## 2.2 PyTorch

PyTorch is an open-source machine learning library based on the Torch library, primarily developed by Facebook's AI Research lab (FAIR). It can be considered one of the most well-known frameworks alongside TensorFlow. PyTorch is distinctive for its excellent support for GPUs and its use of reverse-mode auto-differentiation, which enables computation graphs to be modified on the fly. This makes it a popular choice for fast experimentation and prototyping. Another strength of PyTorch is its extensive set of libraries and tools, such as TorchVision for computer vision and TorchText for natural language processing. Here are some key factors that may lead to choose PyTorch over other deep learning frameworks:

- **Ease to use** it is celebrated for its Pythonic nature and simplicity, making it particularly appealing to beginners thanks to its intuitive syntax and ease of understanding.
- **Flexibility** its dynamic computation graph feature allows for real-time modifications. This simplifies experimentation and debugging, an aspect highly praised by users. PyTorch's design prioritizes user-friendliness, which is especially advantageous for developing complex, iterative model architectures that require frequent adjustments.
- **Tensor Computation** it provides a comprehensive library for tensor operations that are similar to those in NumPy, but with the added advantage of GPU acceleration. This allows for efficient computation and high performance, especially for large-scale neural networks.
- **Distributed training** it is crucial for managing large-scale deep learning tasks, utilizes PyTorch's ability to train models across multiple GPUs or machines efficiently.

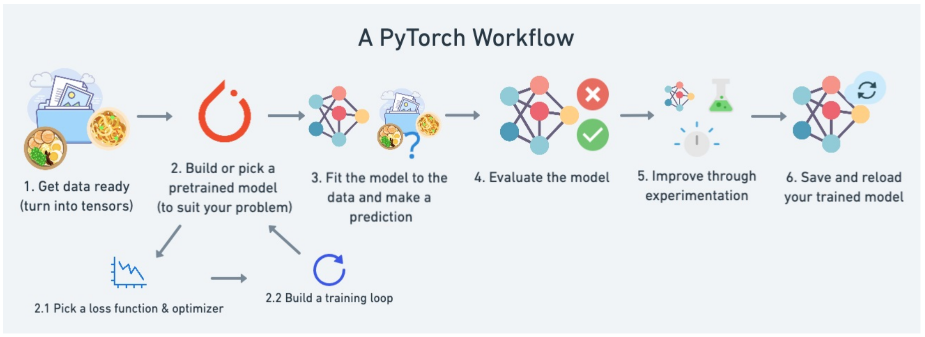

A PyTorch Workflow
1. Get data ready (turn into tensors)
2. Build or pick a pretrained model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
2.1 Pick a loss function & optimizer
2.2 Build a training loop

Figure 1: A standard PyTorch workflow

## 2.3 Keras

Keras is an open-source deep learning framework developed by Google and designed to simplify the creation and training of neural networks achieving at the same time speed and flexibility. Based on principles of user-friendliness, it offers a high-level python API which, by abstracting much of the complexity involved, does not only allow faster prototype and experiment with neural network models but also guarantees robust support for deployment and adoption.

The framework offers the users to choose different approaches to build up model based on their preferences and needs: the sequential model approach, where models are created as straightforward stacks of layers; the functional model API, which allow for the creation of more complex architectures, including multi-input and multi-output models, shared layers, and directed acyclic graphs; and the subclassing approach, which lets the user define its own custom implementation. Additionally, Keras includes a collection of pre-trained models that can be easily loaded and used for various tasks, such as image classification and object detection. This is particularly useful for transfer learning, where these pre-trained models can be fine-tuned on new datasets.

Keras purpose is to provide the users the easiest experience as possible, and therefore it integrates the intuitive and accessible interface with extensive documentation, tutorials, examples, and guides as well as the contribution and assistance from a large and active community in order to support both beginners and experienced deep learning practitioners.

Since becoming part of the TensorFlow project, Keras has benefited from close integration with TensorFlow, allowing user to leverage TensorFlow's powerful features and tools directly within Keras.

### 2.3.1 Keras 3

The Keras 3 update brings significant advancements to the deep learning framework, enhancing its capabilities and usability. One of the standout improvements is the introduction of a unified API -a full rewrite of Keras- that, along with its backend agnostic nature, enables the very same Keras code to run on top of either JAX, TensorFlow or PyTorch backend frameworks. Any Keras model that only uses built-in layers will immediately work with all supported backends. Each user can program on top of the framework that suits its needs the best, and even switch from one to another based on its current goals. This ability to dynamically select the backend allows to potentially obtain the best performance from the framework that best suite each and very combination of model and hardware without any change to the code.

Moreover, Keras 3 improves integration with a broader ecosystem of tools and libraries. It can be used also as a cross-framework language to develop custom components to be integrated into various machine learning pipeline and native workflows in JAX, TensorFlow or PyTorch, for example instantiating model as a Pytorch Module or as stateless JAX function, or exporting them as TensorfFlow SavedModel, thus unlocking unprecedent ecosystem optionality and interoperability.

To better implement such high-level abstraction, Keras 3 is now built as the standalone package 'keras'. Nonetheless, all the precedent versions of the last major update Keras 2 are still available as part of the TensorFlow project, under the TensorFlow Module 'tensorflow.keras'.

Listing 1: Python code snippet illustrating how easy is to set Keras back-end

```python
# Available backend options are: "jax", "tensorflow", "torch".
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
```

In this simple example, it is illustrated how to configure Keras with TensorFlow back-end. This operation should be done before any library imports and utilization.

## 2.4 JAX

In this section, it is analyzed JAX and the features that set it apart from other frameworks.

JAX (Just After Execution) is a Python numerical computing library developed by Google for high-performance machine learning research. It is considered an evolved version of TensorFlow, offering advanced features such as automatic differentiation and vectorized operations. It is similar to NumPy and it has a familiar API. JAX stands out by being able to run on powerful accelerators like GPUs and TPUs while maintaining accelerator-agnostic code. Unlike NumPy, which operates on arrays in an eager, operation-by-operation manner, JAX defines computational graphs of operations and inputs, allowing the compiler to optimize them. This approach significantly enhances performance. It is realized thanks to Accelerated Linear Algebra (XLA) and it represents one of the factors that make this framework so fast. For also this reason, it is not possible to directly update arrays created in JAX. Immutable data structures allow JAX to make aggressive optimizations without worrying about side effects from in-place modifications. Once installed and imported, JAX automatically detects available hardware resources to optimize its computations. While it is possible to manually specify the devices it should use, this is generally not recommended. This feature allows to execute any program in a transparent way of the underlying hardware.
To further understand the benefits of JAX, according to the documentation, we will see in detail the 4 key aspects.

### 2.4.1 Just in time compilation (JIT)

Python is an interpreted language, meaning that every instruction must be translated into machine code at runtime. While this enhances portability and abstraction, it can also create a performance bottleneck, especially for tasks involving many repeated operations. JIT is a technique used to optimize the performance of programs by compiling parts of the code at runtime rather than ahead of time. This means when a function properly marked with @jit annotation is encountered for the very first time during execution, it is compiled "only once". During the initial call, the input array shapes are used to create a computational graph. This graph is then executed by the Python interpreter, with operations recorded step by step. Afterwards, this intermediate representation can be optimized and cached by XLA. Subsequent calls to the same function with identical input array shapes and types will retrieve the cached version, skipping the tracing and compilation steps, and directly accessing the optimized binary blob. To ensure optimal performance, the "jit annotated" function must strictly adhere to the principles of a pure function: it should receive all input data via function parameters and output all results through function returns, avoiding side-effects. By maintaining this structure, the function guarantees consistent results with identical inputs, preventing the need for recompilation due to changes in input types, which can otherwise slow down execution—a counterproductive outcome contrary to our objectives. Hence, it's crucial to avoid altering the types of input parameters.

### 2.4.2 Automatic Differentiation (grad)

In mathematics and computer algebra, automatic differentiation, also called algorithmic differentiation, is a set of techniques to evaluate the partial derivative of a function specified by a computer program. It exploits the fact that every computer calculation, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, partial derivatives of arbitrary order can be computed automatically.
This functionality is particularly useful during the training of neural networks and JAX plays a leading role in that. The self-differentiation mechanism is similar to that present in other libraries: given a Python function with a series of manipulations on tensors, we can automatically obtain a second one that calculates its gradient automatically. JAX supports both forward-mode and reverse-mode automatic differentiation.

- **Forward-mode differentiation** computes the derivative of each intermediate variable with respect to the input variables, while traversing the computational graph in the forward direction. It is efficient for functions with a small number of input variables but can be less efficient for functions with many input variables;
- **Reverse-mode differentiation** computes the derivative of the output with respect to each intermediate variable, while traversing the computational graph in the reverse direction. It is particularly efficient for functions with many input variables and few outputs, which is often the case in machine learning.

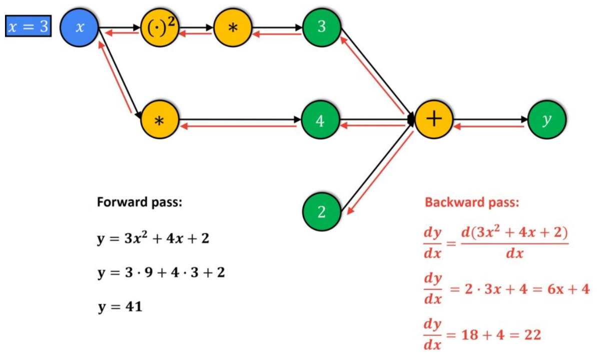

$x = 3$
$x$
$(\cdot)^2$
$*$
3
$*$
4
$+$
$y$
2
Forward pass:
$y = 3x^2 + 4x + 2$
$y = 3 \cdot 9 + 4 \cdot 3 + 2$
$y = 41$
Backward pass:
$\frac{dy}{dx} = \frac{d(3x^2 + 4x + 2)}{dx}$
$\frac{dy}{dx} = 2 \cdot 3x + 4 = 6x + 4$
$\frac{dy}{dx} = 18 + 4 = 22$

Figure 2: Example of automatic differentiation

### 2.4.3 Automatic Vectorization (vmap)

This is a powerful functionality that allows to vectorize a function over multiple inputs, effectively enabling *batch processing*. It transforms a function that operates on individual data points to one that operates on batches of data, without the need for explicit loops. This makes it easier to write efficient, parallelized code that leverages hardware accelerators like GPUs and TPUs. Here is snown a simple example of usage.

Listing 2: Python code snippet using vmap

```python
import jax.numpy as jnp
from jax import vmap

def g(x):
    return jnp.sum(x)

# `vmap` allows `g` to be applied to each row of a matrix:
x = jnp.array([[1, 2], [3, 4], [5, 6]])
result = vmap(g)(x)
# result: DeviceArray([ 3,  7, 11], dtype=int32)
```

In this simple example the function *g(x)* is in charge of summing the elements of a given array as input. This function is passed as argument in vmap function, this means that now it is able to process the data received as input in a vectorial way. Following the SIMD *(Single Instruction, Multiple Data)* pattern, the operation to be executed is fetched and decoded once, then applied at the hardware level to multiple data simultaneously. This approach offers the advantage of avoiding redundant fetch and decode operations for each element in the array. In practice, it eliminates the need to implement software loops, leading to more efficient and streamlined execution. If the underlyng hardware supports vectorial operations the speedup achived can be remarkable in term of speed and memory consumption.

### 2.4.4 Automatic Parallelization (pmap)

This important functionality leverages parallelism in a different way: instead of parallelizing data across the same device applying the same operation on different data, it distributes computations across multiple devices. As the documentation remark, it is mandatory that all the device involved in the computation must be identical (not heterogeneous) e.g. not different GPU models. Under the hood all *jax.pmap* uses XLA to compiles the computation into a series of low-level machine instructions that can be executed efficiently on the underlying hardware and that run on each device simultaneously. These computations are synchronized using a communication protocol to ensure each device has the correct data for its tasks and that the results are accurately aggregated. The protocol employed by *jax.pmap* to consolidate outputs from all devices into a single variable at the end of the computation is known as All-Reduce. After each device computes its local result, it sends this result to a central server. The server then applies the All-Reduce protocol to calculate the aggregate value.

This aggregated result is broadcast back to each device, allowing them to update their local copies of the result. The following example demonstrates a function that benefits from parallelization.

Listing 3: Python code snippet using pmap

```python
import jax.numpy as jnp
from jax import pmap
#  jax.devices() (assuming we have 2 devices)

# Define the function to be parallelized
def square(x):
    return x ** 2

# Create an array of inputs
inputs = jnp.array([1, 2, 3, 4])

# Reshape inputs to distribute across 2 device
inputs = inputs.reshape(2, 2)  # 2 devices, each with 2 inputs

# Parallelize the function using pmap
parallel_square = pmap(square)

# Apply the parallelized function
results = parallel_square(inputs)
# results: [[ 1  4]
#           [ 9 16]]
```

The function square is executed by different devices. The array of inputs is splitted in order to assign evenly balance load to each worker. When the parallel_square(inputs) is called, JAX automatically shards the inputs array into two parts: [1, 2] and [3, 4], and sends each part to a separate device. Each device computes the square of its assigned values. Finally, JAX collects the results from both devices and combines them into a single array.

If the script is tested on a computer which has only a single GPU, to exploit the pmap functionality it is necessary to force the execution of jax using CPU as backend and setting the number of devices equal to the number of core available on your logical CPU.

# 3 Environment Configurations

This project aims to make use of the latest JAX framework and Keras 3. In particular, it aims to thoroughly test and compare the capabilities of Keras 3's cross-framework high-level API across all its backends: PyTorch, TensorFlow, and JAX. Therefore, a well-configured working environment is crucial for this evaluation. To comprehensively assess their performance, the capabilities of Keras 3 backends, as well as standalone JAX, TensorFlow, and PyTorch frameworks will be evaluated in both CPU and GPU environments. Due to resource limitations, tests will be conducted on single-CPU and single-GPU setups.

The project requires the development of a "universal GPU environment" where all necessary backends and frameworks function seamlessly and can utilize the GPU. The procedure for setting up and configuring this environment, both on the Colaboratory and locally, is detailed in the following sections.

## 3.1 Requirements

The first step in setting up the working environment is selecting the appropriate versions of each framework, library, and toolkit. This is particularly important because each framework has very strict compatibility dependencies with each other and with hardware handling platforms like CUDA.

For completeness, the main dependencies of interest are listed in the following tables:

| **PyTorch** | **Torchvision** | **Tensorflow** | **JAX** |
|---|---|---|---|
| $\geq$ 2.1.0 | $\geq$ 0.16.0 | $\geq$ 2.16.1 | $\geq$ 0.4.20 |

Table 1: Keras 3 Compatibility Matrix

| | **Support Platform** | **Linux, x86_64** | **Windows, x86_64** | **Windows WSL2, x86_64** |
|---|---|---|---|---|
| **JAX** | **CPU** | yes | yes | yes |
| | **NVIDIA GPU** | yes | no | experimental |
| **TensorFlow 2.16.1** | **CPU** | Ubuntu 16.04 or newer (64 bit) | Windows 7 or higher (64-bit) | Windows 10 19044 or newer (64 bit) |
| | **NVIDIA GPU** | Ubuntu 16.04 or newer (64 bit) | no | Windows 10 19044 or newer (64 bit) |

Table 2: Compatibility Matrix for different Platforms

| | |
|---|---|
| **WSL2** | Windows 10 (x64 System) version 2004 and higher (Build 19041 and higher) |

Table 3: WSL2 System Requirements

| | **Python version** | **Pip version** |
|---|---|---|
| **Pytorch 2.3** (latest version) | $\geq 3.8$ | - |
| **TensorFlow 2.16.1** | 3.9 - 3.11 | $\geq 19.0$ (Linux) |

Table 4: Software Requirements

| | **NVIDIA GPU with CUDA architecture** | **NVIDIA GPU Driver version** |
|---|---|---|
| **TensorFlow 2.16.1** | 3.5, 5.0, 6.0, 7.0, 7.5, 8.0 or newer | $\geq$ 450.80.02 |
| **JAX** | - | $\geq$ 525.60.13 |

Table 5: System Requirements for GPU support

| | **NVIDIA CUDA Toolkit version** | **cuDNN SDK library version** |
|---|---|---|
| **Pytorch 2.3 (latest version)** | $\geq 11.8$ | $\geq 8.5$ |
| **Tensorflow 2.16.1** | $\geq 11.8$ | $= 8.9$ (not officially supported or tested above) |
| **JAX** | $\geq 12.*$ | $\geq 8.6$ |

Table 6: Software Requirements for GPU support

The final versions of each framework and toolkit used in this project are shown in the following tables:

| | **Google Colab** | **Local Device** |
|---|---|---|
| Python3 | 3.10.12 | 3.10.12 |
| TensorFlow | 2.16.1 | 2.16.1 |
| PyTorch | 2.3.0 | 2.3.0 |
| TorchVision | 0.18.0 | 0.18.0 |
| JAX | 0.4.26 | 0.4.27 |
| Keras | 3.3.3 | 3.3.3 |
| NVIDIA CUDA Toolkit | 12.1 | 12.1 |
| cuDNN SDK | 8.9.6 | 8.9.7 |

Table 7: Frameworks versions used during this project

## 3.2 Colaboratory Cloud Environment

The Colaboratory cloud environment is largely pre-configured due to its intrinsic nature. However, up to the present day it presents a significant issue to carry out this project. Indeed, it comes with TensorFlow version 2.15.0, which is incompatible with Keras 3. Therefore, to update the environment, it is necessary to replace the old TensorFlow and Keras 2 modules with the latest versions. Specifically, installing TensorFlow version 2.16.1 will automatically include the latest Keras 3, eliminating the need for a separate installation.

The following commands, to be executed directly within the Colaboratory cloud environment, address this issue:

Listing 4: python code to update Colab to Keras 3

```
!pip uninstall tensorflow -y
!pip uninstall keras -y
!pip uninstall tf-keras -y
!pip install tensorflow
```

## 3.3 Local Environment

### 3.3.1 System Requirements

The system requirements for developing the desired local environment are as follows:

- Windows 10 (x64 system)

  version 2004 and higher (Build 19041 and higher)

- NVIDIA GPU

  satisfying CUDA architecture and Driver version requirements

### 3.3.2 Preliminary Configuration

First and foremost, it is essential to have a proper NVIDIA GPU to support local computation, since none of the following procedures are meaningful without. This setup is necessary to overcome the limitations of the free Colab account and to make full use of available local resources.

To perform any computation on the local NVIDIA GPU, the NVIDIA CUDA Toolkit must be downloaded and installed on the host machine from the official NVIDIA website:

https://developer.nvidia.com/cuda-downloads

The version of the CUDA Toolkit should be selected based on the operating system and GPU driver version, with the latest version recommended. For this project, CUDA Toolkit 12.4 is installed. The compatibility matrix between CUDA Toolkit and NVIDIA Driver can be found at:

https://docs.nvidia.com/deploy/cuda-compatibility/

### 3.3.3 Windows – JAX – GPU Incompatibility issue

To properly configure a working environment with GPU support, we must consider the available resources and framework compatibility requirements. One significant challenge arises because JAX does not support GPU execution on a Windows machine, which is the only operating system available for this project.

Therefore, a solution needs to be found for this problem. The chosen solution is to utilize Microsoft's Windows Subsystem for Linux (WSL2). According to the official website, WSL2 "is a feature of Windows that allows you to run a Linux environment on your Windows machine, without the need for a separate virtual machine or dual booting." Notably, JAX supports the WSL2 platform. Thus, it is necessary to activate and develop the WSL2 virtual environment. Among the various available distributions, Ubuntu 22.04 is selected as the WSL2 virtual machine for this project. This choice is appropriate because Ubuntu 22.04 comes with Python 3.10.12 pre-installed, which is one of the few versions compatible with all required frameworks (TensorFlow 2.16.1 is compatible only with Python versions 3.9–3.11). On the Ubuntu 22.04 virtual machine, all necessary frameworks and other software can be installed.

The procedures for setting up the WSL2 virtual environment described in this paragraph are adapted from the official Microsoft guidelines, which can be found at:

https://learn.microsoft.com/en-us/windows/wsl/install

If any issues arise, refer to the Microsoft website for detailed information.
To streamline the development of the WSL2 environment and the installation of all required frameworks, a software wizard has been created. A guide for using the `wsl2_project_local_env_wizard` package follows. The subsequent paragraphs provide a comprehensive, step-by-step description of how to manually configure the local environment.

### 3.3.4 WSL2 Local Configuration Wizard

To successfully install a local WSL2 environment with fully functional frameworks and GPU support using the software wizard, follow these steps:

1. **Download the `project_wsl2_local_env_wizard` package from the GitHub repository:**
   - https://github.com/FabioMalloggi/SIC-Machine-Learning
2. **Run the pre-configuration file and apply the changes:**
   (a) Double-click to launch the `wsl2_preconfig.bat` script.
   (b) Grant administrative privileges when prompted.
   (c) Reboot the system.
3. **Run the Windows configuration file:**
   (a) Double-click to launch the `wsl2_config.bat` script.
   (b) Enter your personal account details and password for the virtual environment.

4. **Run the WSL2 Ubuntu environment configuration file:**
   (a) In the command line window that opens, within the Ubuntu-22.04 virtual distribution, run the following commands (this may take some time):

   ```
   cd
   cp /mnt/c/seai_project_temp/project_wsl_conf.sh .
   sh ./project_wsl_conf.sh
   ```

   - Administrative privileges will be requested. If your internet connection is slow, this request may repeat every 15 minutes.
   - The configuration file should reboot the virtual machine at the end. It may take a few minutes to become operational again.

   (b) Close all command line windows.

5. **Run the WSL2 virtual environment:**
   (a) Double-click on the `wsl2_exe.bat` script.
   - This script can be used repeatedly each time you need to access the local environment.

**Additional Steps to Start Programming on a Jupyter Notebook:**

6. **Start a Jupyter Notebook on the host machine connected to the local virtual environment:**
   (a) Run the following command to start a Jupyter Notebook server:

   ```
   jupyter notebook
   ```

   (b) Note the token provided by the server to enable a connection.
   (c) Open a browser on the local host and connect to http://localhost:8888/tree.
   (d) Enter the token in the Jupyter interface on the host browser.

7. **Start a new notebook.**

8. **Begin programming locally with full GPU support.**

#### 3.3.5 WSL2 Local Environment Manual Configuration

This paragraph describes all the steps performed automatically by the local environment wizard. Thus, it should not be executed if the wizard has already installed a perfectly working local environment.

**WSL2 Pre-Configuration - Enable Virtual Machine Feature**

WSL2 is a feature of Windows developed by Microsoft and fully integrated within the Windows operating system. However, the default Windows options do not allow running the WSL2 virtual environment. To enable this functionality, two features must be activated in Windows beforehand:

- Virtual Machine Platform
- Windows Subsystem for Linux

These features provide the local machine with the virtualization capabilities required to use WSL2. They can be found and enabled under the "Turn Windows Features On and Off" subsection of Control Panel. Note that a reboot is needed to apply changes. The Local Environment Wizard performs this through the corresponding PowerShell commands, which require administrative privileges.

**WSL2 Configuration**
There are multiple ways to install WSL2, such as through the Microsoft Store application or by running a few commands in the PowerShell command window. The latter method is used by the Local Environment Wizard and will be explained here. All subsequent commands must executed in a PowerShell command window on the host Windows operating system.

To install WSL and update the Linux Kernel package to the most recent version, set version 2 as the standard version and then download and install the Ubuntu 22.04 distribution, run the following commands:

```
wsl --update
wsl --set-default-version 2
wsl --install -d Ubuntu-22.04
```

During this process, you will be prompted to set an administrator username and password for the virtual machine. Once the command completes successfully, the WSL2 virtual environment will open in the current PowerShell command window. These same commands are performed by the Local Environment Wizard.
To access the Ubuntu 22.04 distribution again:

```
wsl -d Ubuntu-22.04
```

**Ubuntu 22.04 Distribution Configuration**
Once Ubuntu 22.04 is correctly accessed and working, the project-required frameworks and support software must be installed. The Local Environment Wizard includes all subsequent commands in a single script to save time and reduce errors. All subsequent commands must be executed within the WSL2 Ubuntu-22.04 distribution.
First, update the system:

```
sudo apt update
sudo apt upgrade -y
```

Next, install Python (it should be already there) and pip:

```
sudo apt install python3.10
python3 --version
sudo apt install python3-pip -y
pip3 --version
```

To install the GPU-supported version of some frameworks, the NVIDIA CUDA Toolkit must be installed beforehand. Due to compatibility requirements, version 12.1 is needed. The following commands are adapted from the official website:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-
    ubuntu/x86_64/cuda-wsl-ubuntu.pin
sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin
    -600
wget https://developer.download.nvidia.com/compute/cuda/12.1.1/
    local_installers/cuda-repo-wsl-ubuntu-12-1-local_12.1.1-1_amd64.deb
sudo dpkg -i cuda-repo-wsl-ubuntu-12-1-local_12.1.1-1_amd64.deb
sudo cp /var/cuda-repo-wsl-ubuntu-12-1-local/cuda-*-keyring.gpg /usr/
    share/keyrings/
sudo dpkg -i cuda-repo-wsl-ubuntu-12-1-local_12.1.1-1_amd64.deb
sudo apt-get update
sudo apt-get -y install cuda
```

Update the PATH variable to access the CUDA toolkit:

```
export PATH=/usr/local/cuda/bin:$PATH
```

You may want to include this and all subsequent commands to update the PATH variable in the ./bashrc file so that they are applied once for all subsequent sessions. Now, install TensorFlow with GPU support:

```
python3 -m pip install tensorflow[and-cuda]
```

Update the PATH variable for TensorFlow:

```
export CUDNN_PATH="$HOME/.local/lib/python3.10/site-packages/nvidia/
    cudnn"
export LD_LIBRARY_PATH="$CUDNN_PATH/lib:/usr/local/cuda12.1/libnvvp"
export PATH="$PATH:/usr/local/cuda12.1/bin"
```

Next, install PyTorch with GPU support:

```
pip3 install torch torchvision torchaudio --index-url https://download.
    pytorch.org/whl/cu121
```

Finally, install JAX with GPU support:

```
pip install -U "jax[cuda12_pip]" -f https://storage.googleapis.com/jax-
    releases/jax_cuda_releases.html
```

Additionally, the Local Environment Wizard installs Jupyter Notebook to provide an environment where the user can start programming immediately.

# 4 Hands on

This section outlines the key modifications made to the existing Notebooks from the Samsung Innovation Campus Machine Learning course. It includes the renaming of all sub-directories in the folder SIC_ML_Coding_Exercises based on the names and numbers of the contained notebooks to prevent future misunderstandings. The total number of folders is 8 and each one contains several notebooks dedicated to a specific macro-topic. For simplicity, these folders are considered separately in two distinct categories:

1. **Python and Machine Learning** it refers to the first 6 folders containing a basic introduction to Python and a gradual, in-depth analysis of machine learning techniques
2. **Deep Learning** it refers to the last 2 folders containing an introduction to deep learning techniques and the most widely used advanced deep learning networks

In addition to the material already present, two other notebooks are added to better analyze in practice the differences between the various frameworks for training neural networks:

1. **JAX_Custom_Tutorial.ipynb** it contains practical custom implementations of the key features of JAX introduced in previous theoretical section, measuring the performance in term of execution time respect to other standard approaches. This Notebook is well commented and self-explainable. It shows how JAX outperforms among the other in the features considered. For this reason it is not treated in detail in this document.
2. **LeNet5_Frameworks_Comparison.ipynb** it contains an implementation of a famous CNN on the four different frameworks previously analyzed. It will be take into account in detail in the last subsection of this chapter.

All the material can be found at GitHub link:

`https://github.com/FabioMalloggi/SIC-Machine-Learning`

## 4.1 Basics - Python and Machine Learning

The SIC_ML_Chapter_01_Coding_Exercises folder is about how Jupyter Notebook and Python language work. It teaches how to handle basic and complex data types and what are the main operations on them. It contains explanation about Object Oriented programming and the instructions needed to performs loops and conditions. In the final part it also explain how to handle files in file system and how to interact with textual file (Word,Excel,Pdf) formats. In this chapter the only fixes needed are related to deprecated usage of PdfFileReader, updated with the new working version and the commands to interpret of Unicode charathers as integers.

The SIC_ML_Chapter_02_Coding_Exercises folder is an introduction to NumPy and Pandas libraries, largely used for data analysis and data manipulation. It shows how to perform the basic and complex operations on data working with linear algebra and arrays. In the last part, it treats about how to plot the data thought Matplotlib and Seaborn libraries analyzing different kind of plots like Bar, Histogram, Scatter and HeatMaps. The only fixes required are related to missed specification of numerical attributes working on Pandas. Infact, there were present several mathematical operation on non numerical attributes that generated errors. It also required some attributes modification on Seaborn visualization function calls to newer version.

The SIC_ML_Chapter_03_Coding_Exercises folder is a complete overview of Math and Statistic useful to deeply understand how machine learning and neural networks works. It treats topics like discrete and continuous probability distribution, interval estimation and how to performs statistic tests. All is accompanied by graphical representation of the curves and the interpretation of the various shapes motivating the behaviour of particular formulas. The fixes required are minimal and about to specify the type of shape to be considered during the execution of Pearson's test.

The SIC_ML_Chapter_04_Coding_Exercises folder is an introduction to machine learning techniques. It starts explaining how unsupervised methods like KMeans works, passing to dimensional reduction techniques and concluding with linear and customs regressions on real and simulated data. The problem in this chapter is that when executing linear regression on real data, some of the dataset originally proposed were removed due to ethical problems. So to make the Notebooks work, a different public dataset is selected adjusting all the parts of code that referenced to specific attributes.

The SIC_ML_Chapter_05_Coding_Exercises folder continues the journey on machine learning explaining the supervised learning techniques. It starts with basic classification tasks using numeric data, experimenting methods like Naive Bayes, K-Nearest-Neighbour (KNN), Support Vector Machine (SVM) and continues with advanced an more sophisticated ones like Tree Algorithms and Ensemble methods (Voting and Bagging). The modifications needed for this chapter are related to data visualization due to changes in Seaborn API calls. All the other functionalities work very well.

The SIC_ML_Chapter_06_Coding_Exercises folder contains various Notebooks about the usage of strings in real world problems. In the first Notebooks there are explainations on regular expressions and raw string manipulations. Afterwords, it is illustrated a method to do web scraping using BeautifulSoup, a famous library to perform web requests in Python. Finally, there is an overview of Natural Language Processing and how to use NLTK library to manipulate sentences in real world cases. There are also referneces to Information Retrieval techniques and evaluation methods to determine the similarity of a sentence in a predefined context. The fixes here are several:

1. The 0605 notebook explicitly tries to scrape information from Twitter, however in recent years as Twitter.com became X.com, it closed its public API though web scraping and most of the informations are not possible to be scraped without login. For this reason it is left as it is without any modification.

2. There were some method deprecated in the NLTK library so they are updated to current version of the API.

3. The method for obtaining English stopwords has changed in the new version of NLTK, so it has been adjusted accordingly

In this folder there were some .py files about Computer Vision tasks using CV2 python library. This allows to manipulate images with operations such as convolutional and morphological filtering, binaray and adaptive thresholding. These files have been adapted and converted into Notebook files, in order to be easly tested and to be conformed to the others.

## 4.2 Advanced - Deep Learning

The SIC_ML_Chapter_07_Coding_Exercises is an introduction to the framework TensorFlow. It contains exercises to explain the basical mathematical operation on tensors. It also contains an introduction to deep learning classification tasks like recognizing hand written digits using softmax function emphasizing the training, validation and test phases. It is present an interesting insight on TensorBoard tool that allows to see tensors and deep networks structures graphically given the possibility to store it into a log file format. It is also present a detailed explanation of how to train and test a CNN, highlighting the choice of the number of parameters involved in the training on gray scale and coloured images. In the last part, there are references to how applying dimensional reduction thought the neural network "auto-encoder".

The original notebook were written to be executed using TensorFlow 1.0. Due to the introduction of Tensorflow 2.0, the authors adjusted this notebooks to be compatible with the newer version. However, some of the methods used in the current version 2.16 are now deprecated and if they still run maybe they are not optimized. For these reasons, all these notebook are revisited and rewritten according to TensorFlow 2.16. The main fixes are the following:

- removing all the session and placeholder references; sessions are no longer supported as of TF 2.0, as well as placeholder and all sessions' related features; indeed, TF 2.0 prefers functions over sessions and integrates better with the Python runtime with Eager execution enabled by default along with tf.function that provides automatic control dependencies for graphs and compilation
- separating the model, training and loss definitions into properly annotated functions enhances readability and can improve performance if annotated correctly. The training is made into a train_step function decorated with @tf.function (to produce and to be executed as a graph, just as the training session was in TF 1.0) nested within a loop over epochs. GradientTape() function is used to track and better compute derivatives along the generated graph.
- updating linear algebra operations to the most recent API version
- adapting TensorBoard to work with TensorFlow 2.16 due to the removal of Sessions; updated and tested new tensor visualization features, including file logging
- changing datasets import location from tensorflow.keras.dataset to the equivalent version in keras.dataset to prevent future issues with TensorFlow, considering the significant migration of most functionalities to Keras;
- particular handling of the 0708b Notebook, instead of building a model by set of functions, a class implementing tf.Module is defined; this is

also made because the command tf.saved_model.save supports saving tf.Module and its sub-classes, then all weights and biases are defined as instance variables of the Model class. The actual model implementation is defined within ___call___() function of the same class. All class functions (except ___init___()) are also decorated with tf.function since only tf.Variable attributes, tf.function-decorated methods, and tf.Modules found via recursive traversal are saved by tf.saved_model.save.

The SIC_ML_Chapter_08_Coding_Exercises concerns most sophisticated deep neural networks

The Notebooks 0801 and 0802 are about the implementation of fully connected neural networks characterized by low number of layers and neurons. The data used in these exercises are low dimensional and predominantly numeric.

The Notebooks 0803 and 0804 focus on implementing a CNN to recognize digits in gray scale and color images, respectively. The complexity of the neural network is higher compared to the first two notebooks, as well as the type of data processed (images up to 32x32 pixels).

The Notebooks 0805 and 0806 contains implementations of RNN: the first for the recognition of data time series and the second for the a document classification task using a LSTM network.

The Notebooks 0807 and 0808 are related to custom and more complex combination of neural networks to perform Document classification.

The Notebooks 0809 instead perform word embedding using Gensim Word2Vec and pretrained model, doing predominantly inference operations.

The performance analysis of the frameworks is evaluated only on this folder because it represents the most comprehensive one. All the notebooks were already functioning correctly, so the only modification made is to add a code snippet at the start of each notebook that allows the user to switch between different Keras backends.

For more information on the changes made to each notebook, please refer to the commits in the GitHub repository at file:

Fixes_on_SIC_ML_TensorFlow_Review_Version.txt

## 4.3 Custom comparison: CNN - LeNet5 on all frameworks

In this section, a well-known deep convolutional neural network architecture is employed to address a specific classification problem. This network, called LeNet-5, is one of the pioneering structures proposed for computer vision classification tasks. Its simplicity and intuitive design make it an excellent starting point for beginners learning about convolutional neural networks (CNNs). For this problem, a variant of the classical LeNet-5 architecture is employed, utilizing the Fashion MNIST dataset, a widely recognized dataset comprising 28x28 greyscale images of various clothing items. The objective is to classify these images into 10 different categories of clothing.

The network structure is organized in 7 layers:

1. First Convolutional layer: it receives as input a 28x28 single channel image and it consists of 6 filters of size 5x5; it comprehends bias addition and a ReLu activation function;

2. First Average pooling layer: the output is downsampled using average pooling with a kernel size of 2x2 and a stride of 2;

3. Second Convolutional layer: the output from the first pooling layer is passed through a second convolutional layer with 16 filters of size 5x5 and it still comprehends bias addition and a ReLu activation function;

4. Second Average pooling layer: the output is downsampled again using average pooling with a kernel size of 2x2 and a stride of 2;

5. First Fully Connected Layer: the output from the second pooling layer is flattened into a 1D vector to prepare it for the fully connected layers;

6. Second Fully Connected Layer: the flattened vector is passed through a fully connected layer with 120 units;

7. Third Fully Connected Layer: finally, the output from the previous layer is passed through a third fully connected layer with 10 units equal to the number of classes to be predicted. The raw output logits are obtained from this layer, representing the predicted scores for each class. This network does not apply a final activation function (like softmax) because it is applied during the loss calculation or prediction phase to convert logits into probabilities across classes.

In all the three last fully connected layers, a bias vector and a ReLu activation function are also considered.

To perform coherent comparisons the hyperparameters used in the problem to compare the various frameworks are the following:

- Same training, validation and test sets with same split percentages
- Batch size of 1024
- Number of epochs equal to 50
- Adam optimizer with starting learning rate of 0.005
- Categorical Cross Entropy as loss function

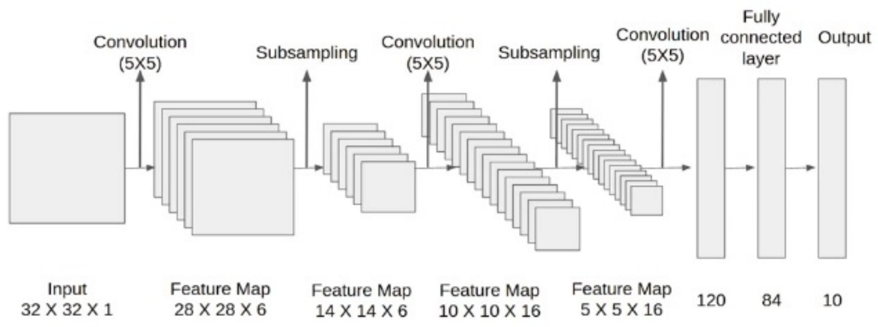


Figure 3: Original LeNet-5 structure

It is possibile to use Flax, an open-source Python library specifically designed for building neural networks on top of JAX, to implement the custom LeNet5 in JAX. It is made up of loosely coupled libraries and its functional API radically redefines what modules can do via lifted transformations like vmap, scan, etc, while also enabling seamless integration with other JAX libraries. The idea is to define each layer of the neural network as pure function, in order to fully exploit the JAX features. Flax defines layers as flax.linen.Module, acting as an immutable dataclass. It can combine sub-module definition and their call via nn.compact. Parameters are stored as immutable Pytrees, a tree-structured container a sort of nested dictionaries. Optax library is used for gradient descend optimization. To integrate model execution, gradient computation, optimizer steps, and leverage Just-In-Time compilation, Flax utilizes its flax.training sub-library. Here, the *TrainState object* plays a crucial role, facilitating operations like function transformations (e.g., jax.grad, jax.jit). However, Flax also has some drawbacks. It typically incurs higher code overhead compared to other frameworks, and its user-friendliness and robustness may not match that of frameworks like TensorFlow or PyTorch. Additionally, while Flax has a growing community, it is considered smaller in size compared to these more established frameworks.

## 5 Results and performances

The performance results are obtained using the following hardware devices:

| **Device** | CPU |
|---|---|
| **Model** | Intel® Core i7-8700 Processor |
| **Cores** | 6 |
| **Threads** | 12 |
| **Clock** | Up to 3.20 GHz |
| **Max Turbo Frequency** | 4.60 GHz |
| **Cache** | Level 1: 384 KB<br>Level 2: 1.5 MB<br>Level 3: 12 MB |
| **RAM** | 16 GB DDR4 2133 MHz |

Table 8: CPU Specifications

| **Device** | GPU |
|---|---|
| **Model** | NVIDIA GeForce RTX 3070 Laptop GPU |
| **CUDA Driver Version** | 12.5 |
| **Runtime Version** | 12.4 |
| **CUDA Capability Major/Minor Ver. No.** | 8.6 |
| **Total Global Memory** | 8192 MB |
| **Multiprocessors** | 40 |
| **CUDA Cores per Multiprocessor** | 128 |
| **Total CUDA Cores** | 5120 |
| **Max Clock Rate** | 1560 MHz (1.56 GHz) |
| **L2 Cache Size** | 4194304 bytes |
| **Max No. Thread per Multiprocessor** | 1536 |

Table 9: GPU Specifications

Training execution times are measured using the Python *time* library. By placing two calls to this function around the train method, the duration of the training phase is recorded. For training instances that take less than 2 minutes, the measurements are repeated 10 times and the average duration is calculated. To ensure accuracy, the computer is dedicated only to this task, with all other processes and activities terminated prior to measurement. All the hyper-parameters in all the tested notebooks like number of epochs, batch size, number of layers and neurons are left invariant as in the original version of the notebooks, in order to preserve the authors work. Of course, the aim here is to watch the different performances of the frameworks under different conditions. Here is an example of how the measurement are performed:

Listing 5: Python code about measurment method

```
import time

t0=time.time()
my_summary = my_model.fit(X_train, y_train, epochs=n_epochs, batch_size
     = batch_size, validation_split = 0.2, verbose = 0)
t1=time.time()

tot_time=end_time-start_time

print(f"Training time: {tot_time}")
```

## 5.1 SIC_ML_Chapter_08_Coding_Exercises

The notebooks in the SIC_ML_Chapter_08_Coding_Exercises folder are error-free for any chosen backend. The table below reports the training performance measurements for the three different backends, with all values representing the time taken in seconds during training using CPU and GPU. For simplicity, the notebooks are grouped in pairs based on the common topics they address. The Notebook #0809 is excluded from consideration as it focuses on Gensim library, whereas this analysis utilizes Keras.

| | CPU | | | GPU | | |
|---|---|---|---|---|---|---|
| Notebook | TF | PyTorch | JAX | TF | PyTorch | JAX |
| #0801 | 109 | 142 | 35 | 170 | 203 | 129 |
| #0802 | 15 | 3 | 2 | 18 | 6 | 6 |
| #0803 | 279 | 265 | 432 | 17 | 27 | 18 |
| #0804 | 1556 | 1681 | 1440 | 92 | 323 | 135 |
| #0805 | 20 | 22 | 3 | 24 | 38 | 13 |
| #0806 | 245 | 562 | 156 | 86 | 728 | 56 |
| #0807 | 69 | 181 | 48 | 22 | 231 | 21 |
| #0808 | 49 | 110 | 49 | 64 | 133 | 61 |

Table 10: Training performance measurement for various frameworks on CPU and GPU

In the first two notebooks, it is evident that JAX outperforms the other two frameworks on both devices. However, the reason it takes more time on the GPU compared to the CPU is that the problem considered is too simple to fully exploit the parallelism required for a vector architecture.

The next two notebooks increase the complexity of the problem and the amount of data to be handled, considering images and convolutional operations. In notebook 0803, it is evident that JAX performs worse than the other two frameworks on the CPU. This is because JAX is designed to be used on a GPU, where the convolution operations are better suited. In notebook 0804, the problem analyzed is the same but considering data of coloured images 32x32 instead of gray scale images of 28x28; the structure of the network is the same except for a different drop-out coefficient. The training time inevitably increases but in this case TensorFlow performs better on GPU respect to the others. However JAX still performs better then PyTorch for this specific problem and data.

Notebook 0805 clearly demonstrates JAX's superiority on both devices. The problem addressed (time series prediction) and the network structure used are relatively simple, which is why the GPU time is higher than the CPU time. This overhead is typically due to the time required to transfer data between the device and the host. A similar and more impressive result is obtained in notebook 0806, where the problem (document classification with LSTM) and the network structure are not overly complex. Notably, PyTorch performs very poorly on both devices for this specific problem, even worse on the GPU. This highlights the advantage of Keras, as it allows users to switch backends if they find the training time excessive.

The last two notebooks address the same problem (document classification) using different types of neural networks. Notebook 0807 employs an LSTM model, yielding quite good results. TensorFlow and JAX have similar training times, both significantly faster than PyTorch. Notebook 0808 combines an LSTM model with a 1D CNN, increasing the network's complexity. In this case, TensorFlow and JAX remain the better choices compared to PyTorch. However, due to the operations performed and the volume of data used for training, the CPU performs better than the GPU.

Of course, the results may vary numerically when executed on different devices. However, the bottom line is that JAX can be an excellent solution for the majority of problems encountered due to its strong performance, especially on GPUs. Its ability to seamlessly handle complex computations and leverage parallel processing capabilities makes it a robust choice for various machine learning tasks. Additionally, the flexibility to switch backends using Keras further enhances its utility, allowing users to optimize training times and performance based on the specific requirements of their projects.

In the next section are shown the results about training time and testing accuracy obtained on the custom Notebook LeNet5_Frameworks_Comparison.ipynb.

## 5.2 LeNet5_Frameworks_Comparison

In this notebook, there are four implementations on different frameworks of the same custom LeNet-5 architecture introduced above. The first three consist of pure PyTorch,TensorFlow and JAX (Flax). The last one is developed using Keras and tested changing the backend to PyTorch, TensorFlow and JAX respectively. In this additional work, the accuracy metric on test set is also considered to provide a deeper evaluation of the reliability of the trained models.

| **Frameworks** | **CPU Train.Time (sec)** | **CPU Test Accuracy (%)** | **GPU Train.Time (sec)** | **GPU Test Accuracy (%)** |
|---|---|---|---|---|
| PyTorch | 171 | 0.904 | 33 | 0.899 |
| TensorFlow | 225 | 0.835 | 24 | 0.821 |
| JAX-Flax | 242 | 0.898 | **21** | **0.897** |
| Keras-PyTorch | 170 | 0.894 | 56 | 0.891 |
| Keras-TensorFlow | 234 | 0.896 | 29 | 0.897 |
| Keras-JAX | 302 | 0.894 | 26 | 0.895 |

Figure 4: Training and accuracy performance measurement for various frameworks on CPU and GPU on LeNet5 custom architecture

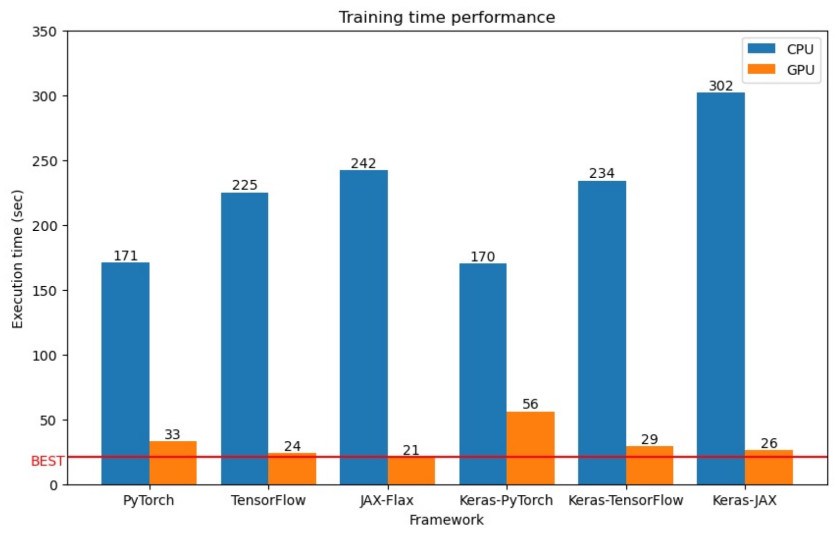


Figure 5: Bar chart visualization of previous training results

In this particular problem, the benefits of using a GPU are quite evident as it significantly reduces training time, making it preferable over using a CPU. The chosen hyperparameters emphasize this aspect and also yield significant improvements in training accuracy. In almost all the cases the accuracy metrics arrives around 90% the only one exception is for pure TensorFlow framework. Of course this accuracy could be improved selecting others hyperparameters or increasing the network structure or increasing the number of data used for training but it is out of this scope. Another interesting aspect is about the speedup obtained with respect to the CPU standard base version, which is defined as the ratio of the time taken by a baseline method (using a CPU) to the time taken by the optimized method (using a GPU):

$$S = \frac{T_{\text{baseline}}}{T_{\text{optimized}}}$$

Here is a table that summarized the speedup obtained for each tested framework:

| **Framework** | **Speedup** |
|---|---:|
| PyTorch | 5.18 |
| TensorFlow | 9.37 |
| JAX-Flax | 11.52 |
| Keras-PyTorch | 3.03 |
| Keras-TensorFlow | 8.06 |
| Keras-JAX | 11.61 |

Table 11: Framework and Speedup Values

In the worst case (Keras-PyTorch), the speedup obtained is 3.03, while in the best case (Keras-JAX), it is 11.6. Overall, it is evident that the pure versions of the frameworks achieve higher speedups compared to their Keras implementations. The execution time is consistently lower in the pure versions than in those using Keras. This difference is due to the overhead introduced by Keras acting as a wrapper. However, it is still a good choice for those who want to achieve a rapid result without losing in details of a specific framework implementation.
Looking to singular frameworks, PyTorch is the one who performs better on CPU but it is the worst performer on GPU (on this specific problem). It is also the one who obtained a little higher accuracy score on test data. TensorFlow can be considered the medium version between PyTorch and JAX on both CPU and on GPU. The pure version of JAX that uses Flax library is the most impactful for performance when using hardware acceleration. However, it is not suitable for use on CPU devices, as it performs worse than other frameworks in this context. This poor performance on CPUs highlights that JAX is specifically designed to execute operations in parallel in a vectorized form, leveraging the underlying hardware acceleration.

## 6 References

1. Introduction to TensorFlow: differences between 1.0 and 2.0
   https://www.tensorflow.org/guide/basics
   https://www.geeksforgeeks.org/tensorflow-1-xvs-tensorflow-2-x-whats-the-difference/?ref=ml_lbp

2. Introduction to PyTorch
   https://en.wikipedia.org/wiki/PyTorch
   https://www.learnpytorch.io/00_pytorch_fundamentals/
   https://datahacker.rs/004-computational-graph-and-autograd-with-pytorch/
   https://pytorch.org/get-started/locally/
   https://github.com/pytorch/pytorch#from-source

3. Key Features of JAX
   https://divyankgarg10.medium.com/about-keras-e36428d51ca8
   https://jax.readthedocs.io/en/latest/
   https://en.wikipedia.org/wiki/Google_JAX
   https://www.educative.io/answers/what-is-jax
   https://en.wikipedia.org/wiki/Automatic_differentiation
   https://www.mishalaskin.com/posts/data_parallel

4. PyTorch discussion on CUDA 12.0 support
   https://discuss.pytorch.org/t/is-cuda-12-0-supported-with-any-pytorch-version/197636

5. What's New in TensorFlow 2.16.1.
   https://blog.tensorflow.org/2024/03/whats-new-in-tensorflow-216.html

6. TensorFlow Installation (PIP)
   https://www.tensorflow.org/install/pip?hl=it
   https://www.tensorflow.org/install/source?hl=it

7. JAX Installation Guide
   https://jax.readthedocs.io/en/latest/installation.html

8. NVIDIA CUDA GPUs and Toolkit 12.1 Download Archive
   https://developer.nvidia.com/cuda-gpus
   https://developer.nvidia.com/cuda-12-1-0-download-archive

9. Microsoft WSL Installation Guide
   https://learn.microsoft.com/en-us/windows/wsl/install

10. Microsoft WSL Manual Installation Guide
    https://learn.microsoft.com/en-us/windows/wsl/install-manual

11. Jupyter Installation Guide and Classic Notebook Installation Guide
https://jupyter.org/install
https://docs.jupyter.org/en/latest/install/notebook-classic.html

12. Keras Documentation and Tutorials
https://keras.io/
https://keras.io/keras_3/
https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras
https://www.coursera.org/articles/what-is-keras

13. Insight on JAX-Flax:
https://github.com/google/flax
https://www.machinelearningnuggets.com/jax-cnn/
https://flax.readthedocs.io/en/latest/
https://phlippe.github.io/media/GDE_Talk_Intro_to_JAX_Flax_2022_
12_06.pdf

14. Insight on LeNet-5:
https://en.wikipedia.org/wiki/LeNet
https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/
https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b