VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF ADVANCED INFORMATION TECHNOLOGY

INTRODUCTION OF MACHINE LEARNING – CSC14005
COURSE PROJECT
NEURAL NETWORKS FOR CLASSIFICATION

—o0o—

INSTRUCTOR(S)

MR. NGUYỄN THANH TÌNH
MR. BÙI DUY ĐĂNG

—o0o—

GROUP'S INFORMATION

No.	Student ID	Student Name	Email	Phone Number
1	22127147	Đỗ Minh Huy	dmhuy22@clc.fitus.edu.vn	
2	22127322	Lê Phước Phát	lpphat22@clc.fitus.edu.vn	0769619867

—o0o—

HO CHI MINH CITY, DECEMBER 2024

TABLE OF CONTENTS

TEACHERS' COMMENTS

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Date: **December, ..., 2024**

Graded and commented Teacher(s)

RESEARCH PROJECT

I. Project Evaluation

a. Work assignment table

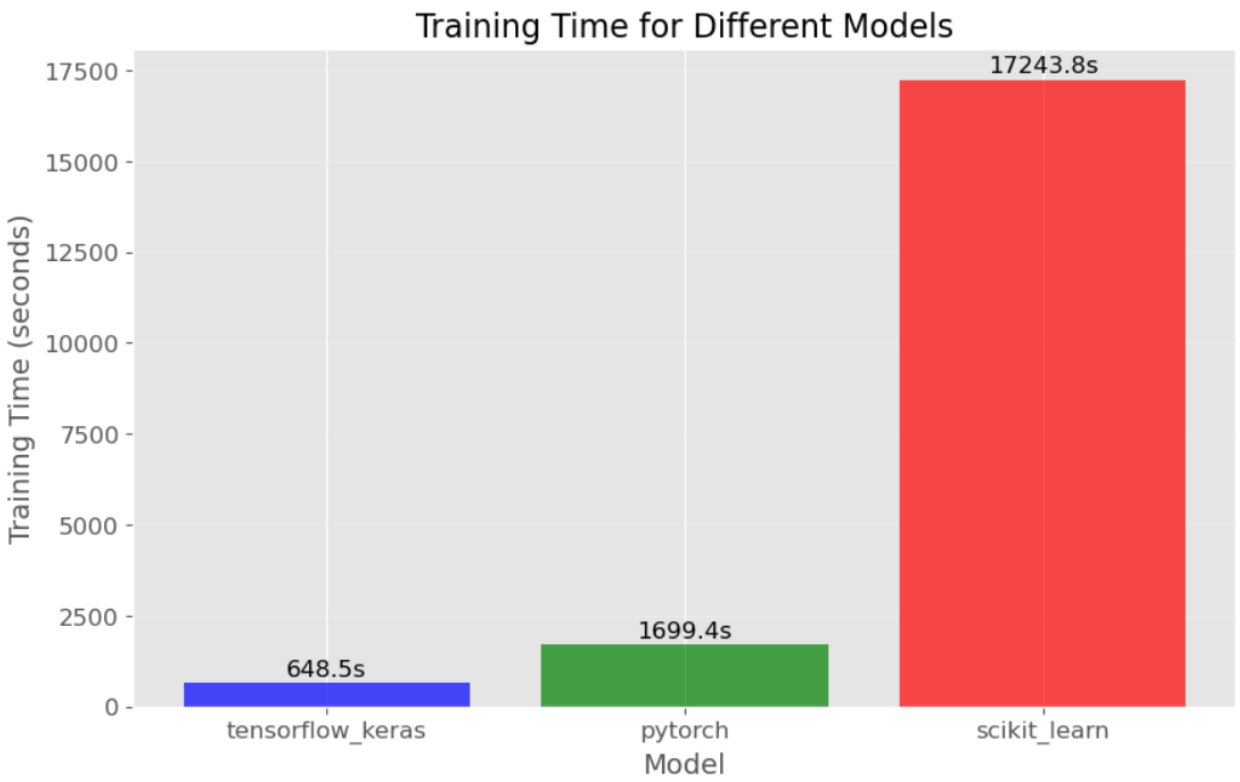
Student ID	Full Name	General Tasks	Detailed Tasks	Completion
22127147	Đỗ Minh Huy	Data Preparation	Data Collection	100%
			Data Preprocessing	100%
			Data Normalization	100%
			Data Flatten	100%
			Data Encoding	100%
		Model Design	Number of layers	100%
			Number of neurons per layer	100%
			Activation Functions	100%
			Drop out or batch normalization	100%
		Selecting Loss Function and Optimizer	Loss Function	100%
			Optimizer	100%
22127322	Lê Phước Phát	Data Preparation	Data Visualization	100%
		Model Training	Scikit-learn	100%
			Tensorflow/Keras	100%
			Pytorch	100%
		Model Evaluation	Each Model Evaluation	100%
			Compare and Analysis	100%
		Futher Usage	Save and Load Model	100%
			Deploy The Model	100%
		Report		100%

b. Self-evaluation of the assignment requirements

No.	Detailed Tasks	Completion Rate
1	Scikit-learn (MLPClassifier)	100%
2	Tensorflow/Keras	100%
3	Pytorch	100%
4	Report	100%

II. Analysis and Comparison of the libraries and frameworks

a. Training Time



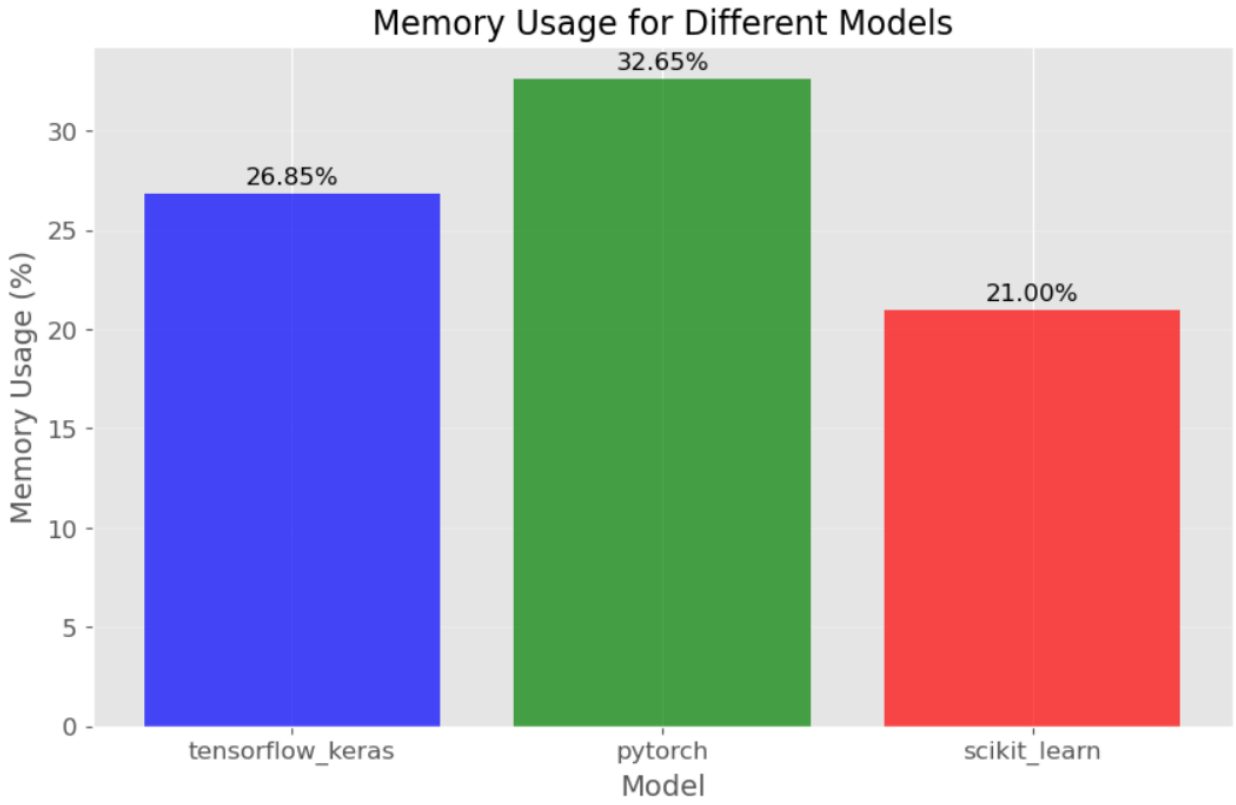
Picture 01. Training Time for Different Models

- ***Scikit-learn***
  - Scikit-learn takes the longest time (17243.8 seconds ~ 4.8 hours), significantly more than Tensorflow/Keras and PyTorch.
  - It is approximately 26.6 times slower than Tensorflow/Keras and 10.2 times slower than Pytorch.
  - Comments: The reason for this high time lies in the fact that
    - ✓ Scikit-learn is designed for traditional machine learning algorithms rather than deep learning.
    - ✓ Scikit-learn operates on CPU only.
    - ✓ It does not support GPU acceleration, which is crucial for deep learning tasks.
    - ✓ It is optimized for smaller datasets, and its inefficiency becomes apparent when handling large datasets or complex computations like deep neural networks.

- ***Tensorflow/Keras***
  - Tensorflow/Keras model has the shortest training time (648.53 seconds ~ 10.8 minutes), indicating it is computationally efficient.
  - Comments: Likely benefits from **GPU acceleration**, efficient computational graphs, and optimizations tailored for deep learning, so make it faster for the larger datasets and neural network training.

- ***Pytorch***
  - Pytorch takes the amount of training time that is 1699.36 seconds ~ 28.3 minutes, which is about 2.6 times slower than Tensorflow/Keras for this task.
  - Comments: Despite being slower than Tensorflow/Keras, Pytorch's **flexibility in dynamic computation graphs** and robust debugging capabilities make it popular for research and experimentation.

b. CPU Memory Usage



Picture 02. Memory Usage for Different Models

- ***Scikit-learn***
  - Scikit-learn has the lowest memory usage (21%) because:
    - ✓ Scikit-learn model is primarily used for traditional machine learning algorithms rather than deep learning issues, so it doesn't rely on large tensors or complex data structures.
    - ✓ Scikit-learn is a library for traditional machine learning algorithms used for clustering, classification, regression, etc., do not require the storage of large intermediate tensors like in Tensorflow/Keras or Pytorch ➔ No need to use large intermediate tensors.
    - ✓ Scikit-learn is designed for efficient CPU usage without requiring complex memory caching or GPU optimizations ➔ Scikit-learn is designed for CPU - only optimization.

- ***Tensorflow/Keras***

- Tensorflow/Keras has moderately high memory usage (26.85%) because:
  - ✓ Tensorflow uses a graph-based execution approach, where a computational graph is built or optimized before execution
  - ✓ The creation and management of intermediate tensors increase memory usage.
  - ✓ When using Tensorflow with Keras API, there is additional overhead due to high-level abstractions provided by Keras.
  - ✓ Tensorflow is primarily optimized for GPUs, and on CPUs, some operations might be less efficient, leading to increased memory usage.

- ***Pytorch***
  - Pytorch has the highest memory usage (32.65%) because:
    - ✓ Pytorch, which is is primarily used for end-to-end building and training of deep neural networks with the ability to create custom models and learning algorithms, uses dynamic execution (eager execution), where operations are executed immediately as they are called. This requires more memory to store the current computation state.
    - ✓ Pytorch has a memory management mechanism that catches temporary memory for better performance, which contributes to higher memory usage.
    - ✓ PyTorch often processes batches at a lower level, leading to the storage of more tensors on the CPU.
    - ✓ PyTorch's more flexible structure compared to Tensorflow can result in higher memory consumption during complex real-time computations.

c. Pros and Cons

- ***Scikit-learn***

  - Pros
    - ✓ Scikit-learn is a widely used open-source machine-learning library for Python.
    - ✓ It's built on top of and integrates with commonly used libraries such as NumPy, SciPy, Matplotlib and Pandas, making it accessible and versatile.

- ✓ It performs well on small to medium-sized datasets with structured features.
- ✓ Besides, it is extremely beginner-friendly with a simple and consistent API.
- ✓ It is usually ideal for quick prototyping and less resource-intensive tasks, which means that we are new to machine learning or developing something using non-neural network algorithms.

▪ Cons

- ✓ Scikit-learn is not optimized for deep learning or image-based tasks, as evidenced by its poor accuracy (44.26%) in this scenario.
- ✓ Training time of models used by Scikit-learn was significantly longer (~ 17243.8 seconds), highlighting its inefficiency for large datasets or tasks requiring complex feature extraction.
- ✓ Scikit-learn doesn't have native support for GPU computing and deep learning. All computations are CPU-based, making it unsuitable for computationally intensive tasks.
- ✓ Scikit-learn is outdated for Neural Network because while it provides tools for neural networks (via MLPClassifier), these are not competitive with modern deep learning frameworks.

⇨ We use it when we require a model for statistical purposes, predictions, classification, or clustering. Scikit-learn works well with relatively small datasets that require general machine-learning computations.

- ***Tensorflow/Keras***

▪ Pros

- ✓ Tensorflow is an open-source framework, which means it is free of cost and can be used by anyone without having to lock into a contract with a vendor beforehand.
- ✓ Keras is the official high-level API of Tensorflow. This allows users to add high-level functionality to their code and reap the benefits of Keras' simplicity. Tensorflow is currently more widely used than PyTorch.
- ✓ Tensorflow/Keras achieved a reasonable balance of training time (~ 648.5 seconds) and accuracy (52.62% in this scenario).

- Cons
  - ✓ It requires substantial hardware resources (e.g., GPUs) for training deep learning models effectively.
  - ✓ Tensorflow may be overkill for simpler machine learning tasks compared to lighter libraries like Scikit-learn.

⇨ We will use Tensorflow/Keras if we require greater funtionality and performance on large datasets and want a library compatible with various coding languages such as C/C++, Java, JavaScript, Go, etc, because it also has extensive multi-platform support.

- ***Pytorch***

  - Pros
    - ✓ PyTorch is a deep learning software library for Python, C++ and Julia. PyTorch is primarily used for end-to-end building and training of deep neural networks with the ability to create custom models and learning algorithms. Essentially, Pytorch is written in Python, which makes it easy for developers of the Python programming language to adapt.
    - ✓ Pytorch's data parallelism is extremely effective as it enables users to split data into batches and send them to various GPUs for processing By employing this method, PyTorch is able to transfer a sizable portion of the workload from the CPU to the GPU. Pytorch's structure is extremely simple and intuitive. It helps implementers easily fix bugs. It is considered one of the easiest deep learning packages to learn.
    - ✓ Dynamic computation graphs make it easier to experiment and debug models compared to TensorFlow's (now optional) static graph.
    - ✓ PyTorch delivered the highest accuracy (57.15%) in this scenario, showcasing strong performance.
    - ✓ At the same time, Pytorch's execution time is quite short, making it easy to train for large datasets and complex neural network structures.
  - Cons
    - ✓ It is difficult to visualize data during model training, users must use a third tool.
    - ✓ For the Pytorch framework, there are no libraries or frameworks that support providing available models through APIs, we have to build and use third-party software to train the model.

- ✓ In this case, Pytorch had a longer training time (~ 1699.4 seconds), indicating possible inefficiencies in certain scenarios.
- ✓ Although improving, Pytorch's ecosystem and development options are still not as robust as Tensorflow's.

⇨ We will use Pytorch if we are developing applications that have computationally expensive tasks, such as natural language processing, computer vision, etc.

d. Ease of Use

- Model selection and training is also quite difficult. We must carefully and carefully choose the model architecture to achieve the highest accuracy.
- At the same time, preprocessing the input data set is also very important because if done wrong, it will lead to reduced model accuracy.
- For the MLP model, all 3 models using the 3 frameworks Skikit-learn, Tensorflow/Keras, and Pytorch all give very low accuracy (all below 60%), because when they We process data by flattening the data, we lose the correctness of the image, break the image structure and MLP considers each pixel as an independent feature, ignoring the relationships between neighboring regions in the image. To overcome this we need to use the CNN model combined with MLP (ANN).
- Directions for export improvement:
  - Using the Convolutional Neural Network (CNN) model
  - Apply Data Augmentation: increase the size of the original dataset by creating different variations of the original image.
  - Use popular model architectures such as VGG16 (or VGG19), ResNet, DenseNet, ...
  - Hyperparameter tuning

REFERENCES

During the process of researching and implementing the ***Course Project: Neural Networks for Classfication***, our team used and referenced some of the following open and electronic documents below:

Programming

[1] How to Configure the Number of Layers and Nodes in a Neural Network (last updated date: 23/12/2024)

Report

[2] Scikit-learn vs. TensorFlow vs. PyTorch vs. Keras (last updated date: 27/12/2024)