OULUN AMMATTIKORKEAKOULU

Joni-Heikki Hermanni Tauriainen

COMPARATIVE ANALYSIS OF MACHINE LEARNING PERFORMANCE: CONVENTIONAL COMPUTER VERSUS SUPERCOMPUTER IN CONVOLUTIONAL NEURAL NETWORK TRAINING

COMPARATIVE ANALYSIS OF MACHINE LEARNING PERFORMANCE: CONVENTIONAL COMPUTER VERSUS SUPERCOMPUTER IN CONVOLUTIONAL NEURAL NETWORK TRAINING

Joni-Heikki Hermanni Tauriainen
Master's thesis
Spring 2024
Data Analytics and Project Management
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Data Analytics and Project Management

Author(s): Joni-Heikki Hermanni Tauriainen
Title of thesis: Comparative Analysis of Machine Learning Performance: Conventional Computer Versus Supercomputer in Convolutional Neural Network Training
Supervisor(s): Manne Hannula
Spring 2024
Number of pages: 60 + 15 appendices

This study examined the difference in running time when a complete Convolutional Neural Network (CNN) learning process was performed on a regular computer versus a supercomputer with data collected by the Oulu University of Applied Sciences (OAMK) 6G studies. The data collection was conducted by an application provided by the Oulu University of Applied Sciences for 6G simulation. The simulation tool is intended to enhance the positioning of the simulated 6G signal transmitter and receiver. This positioning is expected to be accomplished with the assistance of a machine learning script in the future.

In this research, a significant number of teaching image with label pairs for the Convolutional Neural Network (CNN) were collected. This was achieved by running a simulation program and capturing images from the computer screen with a webcam into a Pickle file, which contains the necessary metadata. To ensure the reliability of the collected data, an automated validation system was developed in Python.

The CNN training code was developed in Python using the TensorFlow library. The code was designed to facilitate the modification of the teaching parameters employed. The primary parameters that could be altered were the number of images utilized for training and the pixel size of the image. Additionally, the program prints out indicating the total time required to complete the process, which included the time spent loading the teaching images into memory and the time dedicated to the actual teaching phase. The times were tabulated and subjected to statistical analysis in order to determine the performance of the machines in the given task.

The findings of the research indicate that it is not possible to establish a definitive threshold at which it would be advantageous to transition from the use of a conventional machine to that of a supercomputer in the context of machine learning. This decision is largely contingent upon the availability of resources. However, the study revealed that as the quantity of training data increases, the performance of a standard machine becomes increasingly constrained in the absence of optimization. Furthermore, it was observed that an increasing the pixel size of the training image resulted in an increase in the time required for training. The execution of machine learning scripts places a significant load on the machine, potentially preventing machine from performing other tasks during that time. Therefore, if the execution time is several tens of minutes, it would be beneficial to transfer that work to a supercomputer.

Keywords: Convolutional Neural Network, Supercomputer, Puhti, TensorFlow, Python, 5G, 6G, Machine learning, GFLOPS, PFLOPS

CONTENTS

TERMS AND ABBEREVIATIONS

AI - Artificial intelligence

CNN – Convolutional neural network

FLOPS – One floating-point operations per second

GFLOPS – One billion floating-point operations per second

GUI - Graphical User Interface

PFLOPS – One quadrillion floating-point operations per second

IoT – Internet of Things

1 INTRODUCTION

In recent times, there has been a great deal of discussion surrounding the topic of artificial intelligence, especially with the advent of AI chatbots and image processing AI for consumers, but artificial intelligence is not a new concept. The history of the term artificial intelligence dates to the 1940s. Although the development of artificial intelligence is strongly linked to the development of information technology, it is not easy to define clear steps for the development of artificial intelligence. (Tampereen yliopisto 2023).

The development path has included several phases of stagnation, which are referred to as the winter of artificial intelligence. In that time the development of artificial intelligence has not progressed, there are many reasons for this, such as lack of funding, excessively high expectations, and the poor ability of computers to store and process information. (Tampereen yliopisto 2023).

The importance of artificial intelligence in various industries has grown rapidly in recent years. AI applications have been around for several decades, but the increasing computing power of computers, the development of algorithms and the huge amount of data have only led to several breakthroughs in recent years. With the help of AI, machines can adapt to new situations and perform various tasks, plan and conclude things on an almost human-like level with the information they receive. Using artificial intelligence, machines can independently perform various functions by observing their environment, processing their observations, and making decisions based on that information.

AI can now be found almost everywhere where information technology is present, for example from internet search engines to self-driving vehicles from health care applications to agriculture. (European Parliament 2023).

The current amounts of data may be so large and the algorithms so heavy that at some point, a so-called regular computer is unable or does not make sense to perform machine training calculations. This thesis examines the amounts of data that should be used to switch to using a supercomputer in machine training instead of using a regular computer.

The primary objective of this is to determine through empirical investigation and experimentation, the impact of varying data size on the performance of the computer being utilized for machine learning and AI teaching methods. This investigation aims to identify the data size and processing time at which it is beneficial to transition to a supercomputer for AI learning.

1.1 Telecommunication Technology and its Development Trends

In this thesis, the latest technologies in radio technology from the field of telecommunication technology are an exemplary target, where the aim is to investigate the use of artificial intelligence to guide the direction of the simulated signal transmitter.

1.1.1 Radio Technology

Radio technology is a field of technology that transmits and receives information. The transmission of information occurs at radio frequencies with the transmission of electromagnetic radiation. Radio technology involves many different technologies and applications that enable information processing. The most important element needed in radio technology is the antenna and it is needed in almost every radio technology application. The function of antennas is to transmit and receive different radio signals at different frequencies.

There are different types of antennas for different purposes. Antennas are reciprocal, meaning their characteristics for transmitting and receiving radio signals are the same. For example, if the antenna sends a radio signal in a certain direction, it can only receive radio signals from the same direction. (Räisänen & Lehto 2001, 151).

Radio waves propagate through a medium as wave movements, and the medium and other environmental factors can cause interference in them. A wave can, for example, attenuate, scatter, bend, and reflect off different media and surfaces. For efficient data transmission, a radio transmitter should be capable of producing a strong signal with sufficient power, and the frequency should be as precise as possible, with a spectrum bright enough to avoid interfering with other radio frequency users. Correspondingly, the receiver must then be sufficiently sensitive and selective to be able to distinguish even a weak signal among other radio signals. (Räisänen & Lehto 2001, 195).

1.1.2 5G Network Technology

5G network technology is the fifth generation of the mobile network system. It enables even faster, more reliable, and versatile data transmission than previous network technology generations, up to 10 times faster than the previous 4G generation offers. With 5G, higher data speeds, significant capacity, and low latency have been achieved for data transmission.

Network slicing enables the customization of network services as needed. It has laid the foundation for many new applications and services.

The benefits of 5G include its increased broadband speed (500 Mbps - 1 Gbps), which has enabled the expansion of IoT to various devices. Reliability and low latency allow for remote control and automation usage.

1.1.3 6G Network Technology

6G network technology will follow 5G technology approximately in the year 2030. For 6G networks, radio technologies are being developed to utilize different radio frequencies ranging from over 100GHz to 1THz.

In the future network, telecommunications, observation, and imaging can be used in one device. With 6G's radio waves, it will be possible to image, for example, people and the environment without a traditional camera. (Hintsala 2023).

6G is expected to achieve 50-100 times the capacity of 5G, as 5G needs to support 1 million devices per square kilometer, whereas 6G has been proposed to support up to 10 million devices.

With 6G, we move to higher spectrums, which means that radiation power will decrease. The drawback of higher spectrums is that they do not propagate far without atmospheric and other interferences attenuating them. However, when transmission distances are short, base station transmission powers can be low. This, on the other hand, leads to the need for more base stations than before.

The potential applications of this technology are limitless. According to Nokia, there are six distinct areas of technology: artificial intelligence and machine learning, frequency bands, sensing networks (environmental, human, and object perception), extreme connectivity, new network architectures, and security and reliability. (Nokia 2023).

1.2 Artificial Intelligence and its importance in Telecommunications Technology

The term Artificial Intelligence (AI) is used to describe the ability of computer systems to perform tasks that are considered intelligent. Defining AI is challenging because there is no precise delineation of intelligence. AI is usually understood to refer to machine learning, natural language processing, and decision-making.

In machine learning, a large amount of data to be taught is fed to the computer so that it learns to make a decision according to the task. It learns from the data, and no specific instructions are provided to it for learning and completing the task. It achieves learning by sequentially going through various learning algorithm and phases, which enable it to identify common patterns and features in images, thus evolving step by step. Eventually, it creates a model of the subject matter, with which it seeks to predict a specific outcome. The outcome can and is tested with separate test data. The model becomes more precise the more and different teaching data is available.

In telecommunications technology and networks, artificial intelligence is used for a wide range of purposes because of its ability to learn, adapt, and perform intelligent tasks. It can be used to automate tasks, improve performance and security, as well as offer new services and products. (Heerdan 2023).

In information technology, artificial intelligence can be used, for example, in coding. It can assist a programmer in finding errors in the code, as well as suggesting fixes and optimizations. In addition, it can also to be used to generate ready-made code.

Artificial intelligence is also used to enhance security. It can protect against various types of cyberattacks, such as identifying malware and preventing its spread, as well as detecting and preventing online abuses. An example is the spam filter of email systems.

Artificial intelligence is also used for analyzing network traffic, detecting issues, and optimizing network traffic. It can improve network performance and reliability and reducing costs for administrators. (Aksela, Marchal, Patel, Rosenstedt & WithSecure 2022).

The most widely discussed aspect of artificial intelligence at present is the use of chatbots, which are based on natural language processing. These systems enable users to engage in conversation with computers in a manner that is similar to that of a human-to-human interaction.

1.2.1 The Use of Artificial Intelligence in Optimizing the Performance of Telecommunication Networks

Telecommunication technology networks are constantly developing with new technologies and applications. Their capacity increases, making it possible to support the growing amount of data. Performance improves with development and then they are able to offer faster and more reliable connections.

Because of this, the existing network infrastructure must be updated to meet today's requirements both in terms of software and by renewing physical devices.
Updating physical devices is more expensive than updating software, which has led to a focus on the benefits of artificial intelligence in the use of network technologies. (Juniper 2024; Cisco 2024).

The Artificial intelligence can be utilized managing networks, detect and fixing problems, and optimizing capacity. This can improve performance and reliability and reduce costs. (Seraydarian, Mosinyan & Kotolyan 2023).

The Edge computing is one good example of the help of artificial intelligence. In edge computing, data is processed close to the source of the data, so to speak, in the Edge area of the network and the data is not transferred out of this area. In practice, the Edge is a device that collects data from this area, for example, an IoT device, smartphone, tablet or base station. (Gill 2023; IBM 2024).

This frees up the bandwidth of data transmission by not sending data to a central hub for processing and then returning the processed data. This also results in reduced latency in device operations, fewer network disturbances, and increased efficiency, when data transfer does not consume power. (Gill 2023; IBM 2024).

The importance of edge computing has grown a lot in the proliferation of IoT devices, when at the current pace an incredible amount of network bandwidth would be required to serve all those who need the service. (Zieniūtė 2023).

1.2.2 Convolutional Neural Network

The Convolutional Neural Network (CNN) is one of the areas of artificial intelligence that specializes in identifying and classifying videos and images. CNN uses convolutional operations to identify different features and properties in images. CNN has multiple layers, and each layer contains convolutional layers, pooling layers, and activation layers. In the convolution phase, various filters are used to identify shapes and properties in the image, such as edges, angles, and texture. The pooling phase reduces image sizes and simplifies the features in the image. The activation phase improves the performance of the neural network.

In the learning phase, a sufficiently large amount of training data is needed, meaning images that someone has classified in advance. After that, the neural network examines the images at the pixel level and develops computational parameters based on them, which it uses for image classification. The classification accuracy can be good after just one pass, but usually, multiple convolution and pooling layers are needed. Layers can be built as many as desired, taking learning to a deep level. Once the neural network is trained well enough, it can recognize and classify images it has not seen before.

In the training data, it is good to have a sufficiently large amount and as diverse as possible training data, meaning images. It is beneficial for the object being taught in the image to vary in shape, position, angle, rotation, to obtain a diverse training material. This material generation can also be done programmatically using the TensorFlow library, allowing existing images to be modified into different shapes and positions. (Crabtree 2023; TensorFlow 2024).

After training, the performance is tested on another set of data, the test data, which is also human-classified data but has never been presented in the training phase. The use of separate data allows for the elimination of the phenomenon known as overfitting, whereby the system has learned the images by rote.

When everything is done well, the end result should be so accurate that a person can rely on the machine to handle the task it has been taught, allowing the person to focus on more demanding tasks. (Insta 2023).

The CNN has the potential to be utilized in a multitude of applications, with the scope of its applications limited only by the imagination of the user. CNNs can be employed in a variety of tasks, including the recognition of faces, vehicles, and license plates; the counting of studied objects; and even the control of robots in robotics applications.

1.2.3 The Significance of Computing Power

In comparing the computing power of available computers, the floating-point performance is measured over time. In this thesis, GFLOPS (Giga Floating-point Operations Per Second) is used, indicating how many billion floating-point operations are per-formed per second. In supercomputers, performance is often measured in PFLOPS (Peta Floating-point operations Per Second), which is a million GFLOPS. This involves truly astonishing figures. (Anant 2023).

The work utilizes a straightforward GFLOPS calculator written in Python, as detailed in a later section. It is validated against one LinPack software (Soft for Bro 2024), which performs numerical linear algebra to calculate computer performance. It was developed in the late of 1970s for calculating the performance of supercomputers. (Matlis 2005).

2 PURPOSE AND GOAL OF THE THESIS

The purpose of this work was to experimentally find out on a practical level when the amount of machine learning data size and the time spent on it are so large that it is worthwhile to switch to using a supercomputer instead of a regular computer for machine learning.

The results of the study are beneficial for the development of artificial intelligence and supercomputing research at Oulu University of Applied Sciences, aiding in both research and education. Understanding when it is worthwhile to invest resources in utilizing a supercomputer and the potential benefits it can bring will contribute to the advancement of both research and teaching activities.

The results obtained can also be used by Oulu University of Applied Sciences to further develop the 6G signal antenna simulation tool used in this work. In this work, the image material of a sphere simulating the 6G network transmission signal generated by this simulation tool was used. The purpose was also to gather information on whether the collected material could be sufficient to make a good transition to teaching AI on a supercomputer.

The main focus was to study and find out how the amount of data collected for machine learning and how artificial intelligence training methods affect the performance of the computer used, and at what point it would be good to switch to using a supercomputer to perform teaching.

The thesis sought answers to the following research questions:

- What is the data size at which it becomes advisable to use a supercomputer instead of a regular computer?
- How does the teaching method of artificial intelligence affect the computing time and when it becomes advisable to use a supercomputer instead of a regular computer?

3 IMPLEMENTATION OF RESEARCH WORK

In this work, the initial phase involved a review of the general history and current state of the fields of telecommunications technology and artificial intelligence, along with relevant literature.

After that, a large amount of data was collected in the work, for teaching artificial intelligence, using a special solution developed for simulating the optimization of the communication system. The collected data included images and the related metadata. The images and their related metadata were initially examined manually to ensure their suitability for education, and necessary adjustments were made to the simulation and data transmission methods as needed.

The artificial intelligence teaching algorithm was implemented using Python code once a sufficiently large set of training data had been collected.

The training code for artificial intelligence was designed in a way that allowed for easy variation of different parameter values during the execution of the training process. In this manner, it is possible to easily alter various processing aspects on the executing machine, in addition to handling a large amount of data, using suitable parameter settings. For instance, this can be achieved by adjusting the number of epochs. In the study, the goal was to investigate when it is worthwhile to transition using a more powerful supercomputer.

Both a regular computer and a supercomputer were utilized in the training of artificial intelligence. The execution time during the training process was recorded, and conclusions were drawn to determine when it would be beneficial to transition to using a more powerful supercomputer. Additionally, a computer-intensive FLOPS calculator was developed to provide an indication of the machine's efficiency. The information obtained from this was utilized when comparing the performance of different machines.

3.1 Used Devices

In this thesis, two distinct computers were utilized. One was a laptop equipped with the Windows operating system, while the other was the Puhti supercomputer, situated in Kajaani.

3.1.1 Window Laptop

This work used Lenovo's ThinkPad Window 11 operating system. The machine is equipped with an Inter(R) Core™ i7-8550U CPU @ 1.80 GHz, 16.0GB of RAM, and a base frequency of 1.8 GHz, with a turbo boost frequency of up to 4 GHz. The laptop has 1 CPU and 4 cores. (Intel 2024).

The laptop is not particularly powerful in comparison to contemporary standards, yet it exhibits approximately 70 GFLOPS computing power by running benchmark testing via script wrote for this thesis and running The LinPack software. The manufacturer's specifications indicates a theoretical maximum of 108.8 GFLOPS. (Intel 2024).

3.1.2 Supercomputer

Supercomputers are the fastest and most powerful computers, capable of performing highly complex and intensive computations that are not possible with regular computers. Supercomputers are collections of many highly powerful computers, connected to each other with very fast communication data links. They have multiple processors, a large amount of memory and storage, allowing them to process massive amounts of data in a short time. In general, they are capable of processing millions of billions of calculations per second. Performance is typically reported in PFLOPS. (Lutkevich 2022)

Supercomputers are used many different areas like as weather forecasting, cryptography, scientific research, engineering, artificial intelligence, and machine learning.

The primary challenge associated with supercomputers is their high cost and significant energy consumption, which is further compounded by the generation of considerable heat during operation. To ensure their optimal functioning, these computers require specialized facilities for both construction and operation.

3.1.3 Puhti Supercomputer

Puhti is CSC's supercomputer located in Kajaani, Finland. It is Atos Bull Sequana X400 system. Puhti consist of near 700 CPU nodes with 192 GB – 1.5 TB of memory. Each nodes have two Intel Xeon Gold 6230 -processors with 20 cores. Each core runs at 2.1 GHz. Theoretical peak performance is 1.8 Petaflops and 4.8 PB of storage capacity of Puhti.

Puhti have AI Artificial Intelligence Partition which have 80 nodes with a total peak performance of 2.7 Petaflops. Each nodes have two Intel Xeon Gold 6230 -processors and four Nvidia Volta V100 GPU's. AI part have 384 GB of main memory with 3.2 TB of fast local storage. (CSC 2019).

3.2 Python

In this work, Python was used to develop all the codes. Currently, it is one of the most popular programming languages, distinguished by its readability, abstraction, and advanced libraries. The code written in Python is interpreted, meaning it is not pre-compiled; instead, the code is interpreted at runtime. Its primary limitation is slightly inferior performance. (Vadapalli 2024). Nevertheless, the issue of performance is not a concern in this context when it comes to determining whether machines are capable of executing a given task.

The code written for computers to execute was not optimized, and as such, is the same for both environments. In order to achieve optimal performance, the code must be optimized along with the execution environment, for example, by using parallel computations. This optimization was not within the scope of this particular study; however, it is an important aspect to consider in future studies.

3.2.1 Python libraries and modules

The Python language already has well-developed libraries and modules that will be used in this thesis.

The most important libraries and modules are TensorFlow, scikit-learn, numpy, pickle, cv2, psutil, platform, tkinter and matlotlib.

3.2.2 Python Virtual Environment

The Python Virtual Environment is a tool that enables the creation of distinct Python environments for specific projects. On a virtual environment, it is possible to install specific versions of Python, libraries, and packets that differ from those installed on the host computer.

The sharing of a defined Python virtual environment with other computers is facilitated by the avoidance of conflicts of dependencies in local environments on other computers. Each virtual environment operates within its own sandbox, utilizing the same defined environment settings.

The process of creating a virtual environment is straightforward. To initiate the creation of a new environment, enter the following command into the terminal: *"python –m venv name_of_env".* The name_of_env parameter represents the name of the environment to be created. Once the environment has been created, the command to activate it is *"source env/bin/activate",* which is applicable to both Linux and Mac systems. Alternatively, for Windows users, the command is *"env\Scripts\activate.bat".* To close an environment, one must enter the following command in the terminal: *"source env/bin/deactivate"* (on Linux and Mac) or *"env\Scripts\deactivate.bat"* (on Windows).

To utilize the same Python installation in a different environment, one must first store the current setup by entering the following command: *"pip freeze > name_of_file.txt"* This command saves a list of installed libraries from the current virtual environment. Subsequently, the file created should be transferred to the new virtual environment and the command *"pip install -r name_of_file.txt"* should be executed.

3.3 Collection Training Data

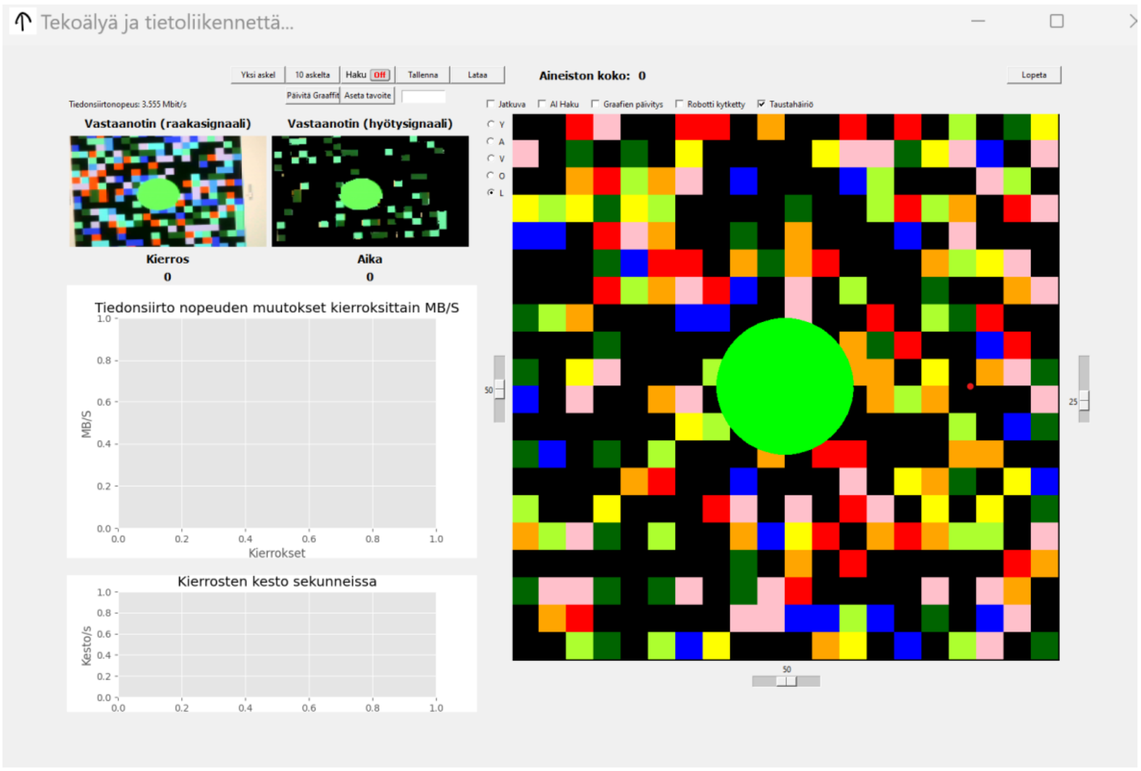
The acquisition of images for training Convolutional Neural Networks (CNN) was accomplished using a pre-implemented Python graphical user interface (GUI) application.

The application is designed to create a graphical simulation of a 6G radio transmission signal, where there may be an obstacle between the transmitter and receiver, as well as random interference signals. The simulated radio signal generates a variety of randomly generated shapes, including a sphere (Figure 1), a hemisphere (Figures 2, 3, and 4), and a blank (Figure 5). The shape of the signal is designed to determine the transmission power when there is an obstacle between the transmitter and receiver. When the sphere is not perfectly circular, it represents an obstacle between the transmitter and receiver. The size of the sphere simulates the distance between the transmitter and the receiver and the resulting change in signal strength. Using machine intelligence, the half-sphere should be able to be guided to move in the right direction to become a whole. For example, Figure 4 illustrates the situation in which the sphere is cut off at the right edge. In this scenario, the antenna must be relocated to the left to achieve the most optimal circular configuration. In the teaching material, different colored rectangular pieces simulating interference signals are also placed between the transmission sphere to simulate, among other things, interference due to reflections in the transmission of data (Figures 1, 2 and 3).

For collecting the training data simulation application was running a long time period and images ware captured at regular intervals by using a webcam. As each image is captured, the application stores it along with relevant data, like shape of sphere, in a pickle file. To manage data efficiently, the size of data collected in a single pickle file was set to approximately 10 MB. The collected data is then dumped into a single file, and subsequent data is stored in new files. This approach ensures that, in the event of any issues, the loss of all collected data is prevented, and facilitates easier data handling in the future. Each pickle file contains around 200 images with accompanying information, with the relevant data for this study being the image and its label.

3.3.1 Collecting Training Data Images

In the data collection process, a simulation application, implemented in the Python language and featuring a graphical interface, was executed. The simulation process could be initiated from the application. The recording of the simulated video stream from the window was accomplished using a webcam. Approximately 200 image captures were taken from the video stream for one pickle file. The application's basic screen is depicted below, with the largest image on the right side representing the image stored by the webcam.


Tekoälyä ja tietoliikennettä...
Yksi askel
10 askelta
Haku Off
Tallenna
Lataa
Päivitä Graaffit
Aseta tavoite
Lopeta
Vastaanotin (raakasignaali)
Vastaanotin (hyötysignaali)
Tiedonsiirto nopeuden muutokset kierroksittain MB/S
MB/S
Kierrokset
Kierrosten kesto sekunneissa
Kesto/s
Y
A
V
O
L
50
25
50

Tiedonsiirtonopeus: 3.555 Mbit/s

Aineiston koko: 0

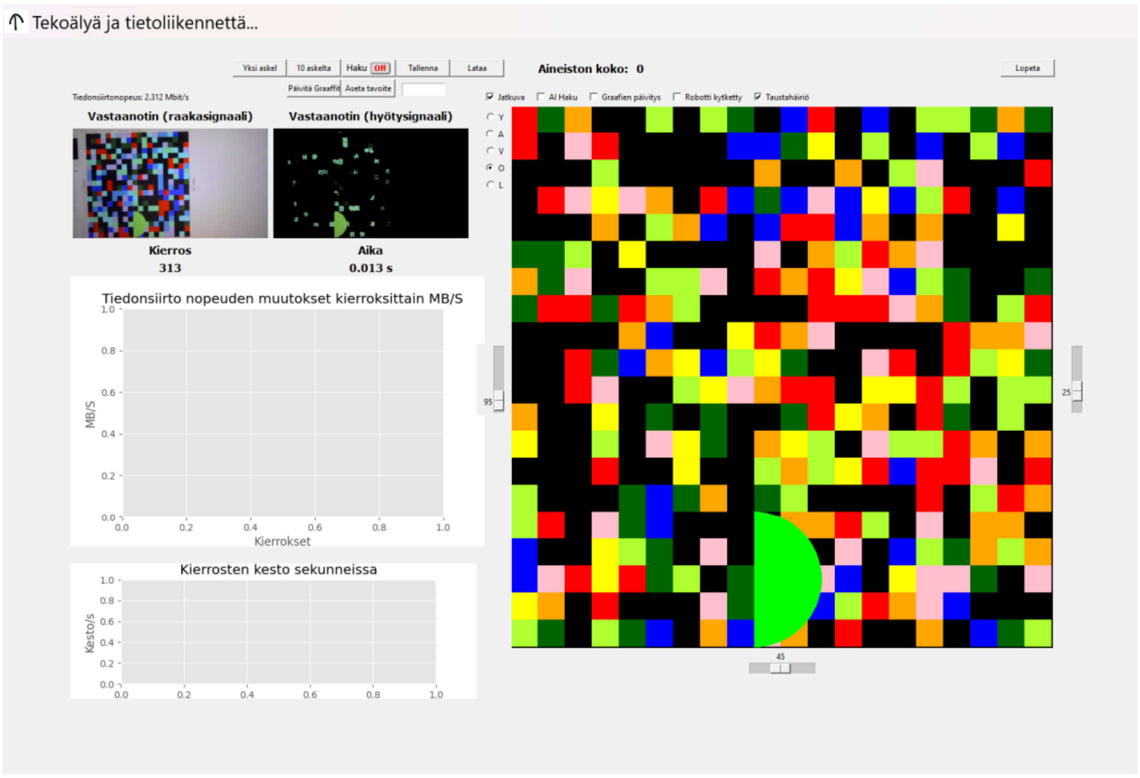
Kierros 0

Aika 0

☐ Jatkuva ☐ AI Haku ☐ Graafien päivitys ☐ Robotti kytketty [✓ Taustahäiriö]

FIGURE 1. Example of a sphere with background noise

Figure 1 provides an example of a simulation tool that illustrates a sphere with background noise. This simulates a scenario in which there are no obstacles between the transmitter and the receiver, yet there is some random background noise that impedes the efficacy of the signal.


Tekoälyä ja tietoliikennettä...
Yksi askel
10 askelta
Haku Off
Tallenna
Lataa
Päivitä Graafit
Aseta tavoite
Lopeta
Vastaanotin (raakasignaali)
Vastaanotin (hyötysignaali)
Tiedonsiirto nopeuden muutokset kierroksittain MB/S
MB/S
Kierrokset
Kierrosten kesto sekunneissa
Kesto/s
Y
A
V
O
L
95
25
45

Tiedonsiirtonopeus: 2.312 Mbit/s

Aineiston koko: 0

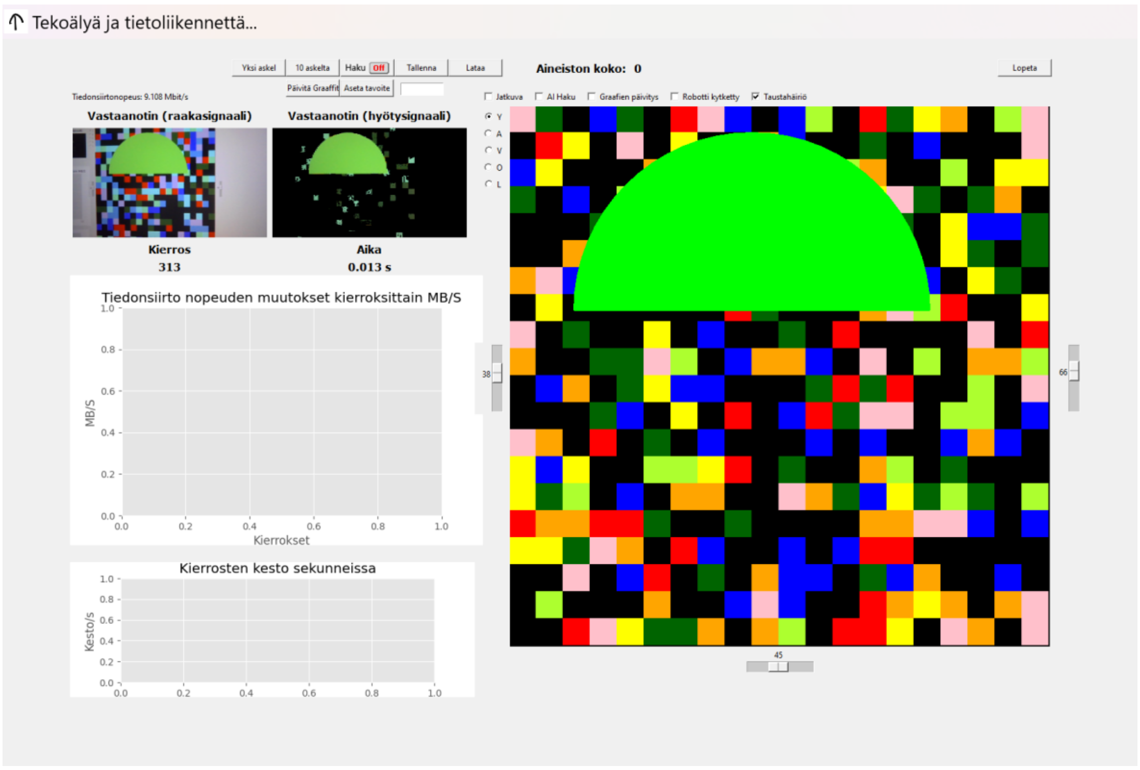
[✓ Jatkuva] AI Haku Graafien päivitys Robotti kytketty [✓ Taustahäiriö]

Kierros 313

Aika 0.013 s

FIGURE 2. Example of half right a hemisphere with background noise

Figure 2 provides an illustration of a simulation tool that depicts a hemisphere with background noise. This simulation scenario depicts a scenario in which an obstacle exists between the transmitter and the receiver, which blocks the left side of the signal. Additionally, some random background noise is introduced, which impairs the effectiveness of the signal.


Tekoälyä ja tietoliikennettä...
Yksi askel
10 askelta
Haku Off
Tallenna
Lataa
Päivitä Graaffit
Aseta tavoite
Lopeta
Jatkuva
AI Haku
Graafien päivitys
Robotti kytketty
Y
A
V
O
L
Vastaanotin (raakasignaali)
Vastaanotin (hyötysignaali)
Tiedonsiirto nopeuden muutokset kierroksittain MB/S
MB/S
Kierrokset
Kierrosten kesto sekunneissa
Kesto/s
38
66
45

Tiedonsiirtonopeus: 9.108 Mbit/s

Aineiston koko: 0

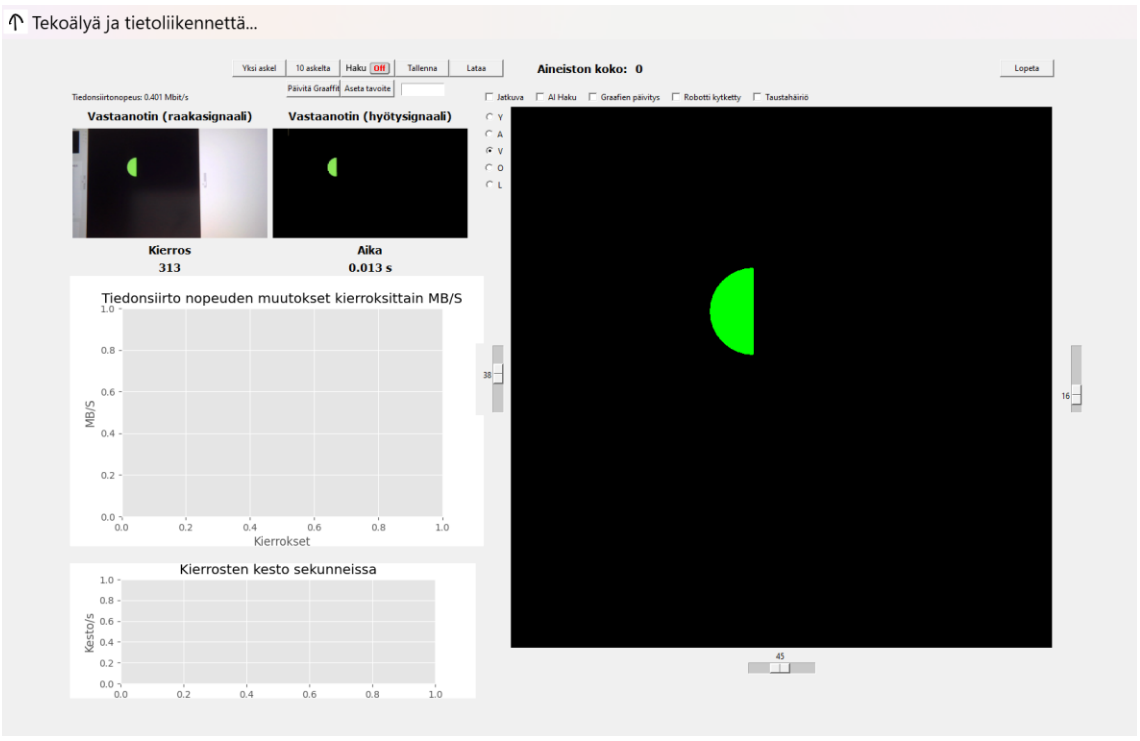
[✓ Taustahäiriö]

Kierros 313

Aika 0.013 s

FIGURE 3. Example of top half a hemisphere with background noise

Figure 3 provides an illustration of a simulation tool that depicts a hemisphere with background noise. This simulation scenario depicts a scenario in which an obstacle exists between the transmitter and the receiver, which blocks the bottom side of the signal. Additionally, some random background noise is introduced, which impairs the effectiveness of the signal. The size of a hemisphere illustrates the case where the transmitter and receiver are close to each other.


Tiedonsiirtonopeus: 0.401 Mbit/s

Aineiston koko: 0

Kierros 313

Aika 0.013 s

Y A [✓ V] O L

Tekoälyä ja tietoliikennettä...
Yksi askel
10 askelta
Haku Off
Tallenna
Lataa
Päivitä Graaffit
Aseta tavoite
Lopeta
Jatkuva
AI Haku
Graafien päivitys
Robotti kytketty
Taustahäiriö
Vastaanotin (raakasignaali)
Vastaanotin (hyötysignaali)
Tiedonsiirto nopeuden muutokset kierroksittain MB/S
MB/S
Kierrokset
Kierrosten kesto sekunneissa
Kesto/s
38
16
45

FIGURE 4. Example of left half a hemisphere without background noise

Figure 4 provides an illustration of a simulation tool that depicts a hemisphere without background noise. This simulation scenario depicts a scenario in which an obstacle exists between the transmitter and the receiver, which blocks the right side of the signal. The size of a hemisphere illustrates the case where the transmitter and receiver have some distance between them.

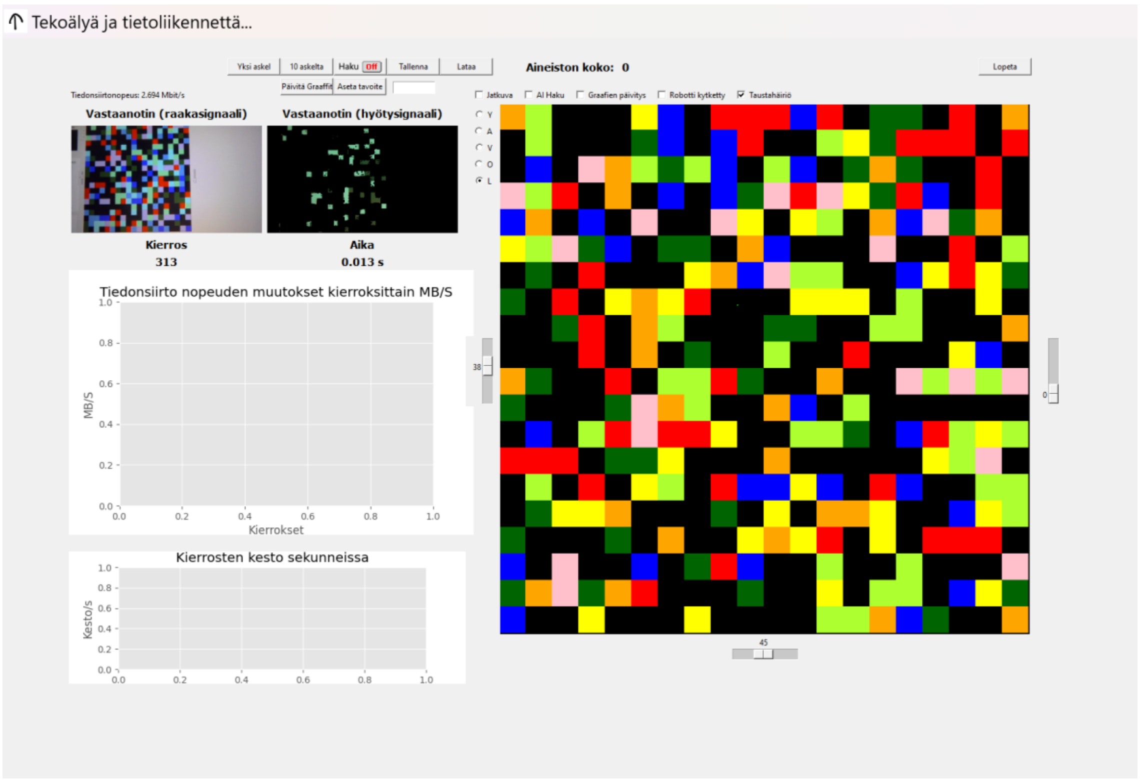


FIGURE 5. Example of missing sphere with background noise

Figure 5 depicts a simulation tool that illustrates a scenario where no signal is visible and there is also background noise. This simulation scenario depicts a scenario where there is a barrier between the transmitter and the receiver that completely blocks the signal.

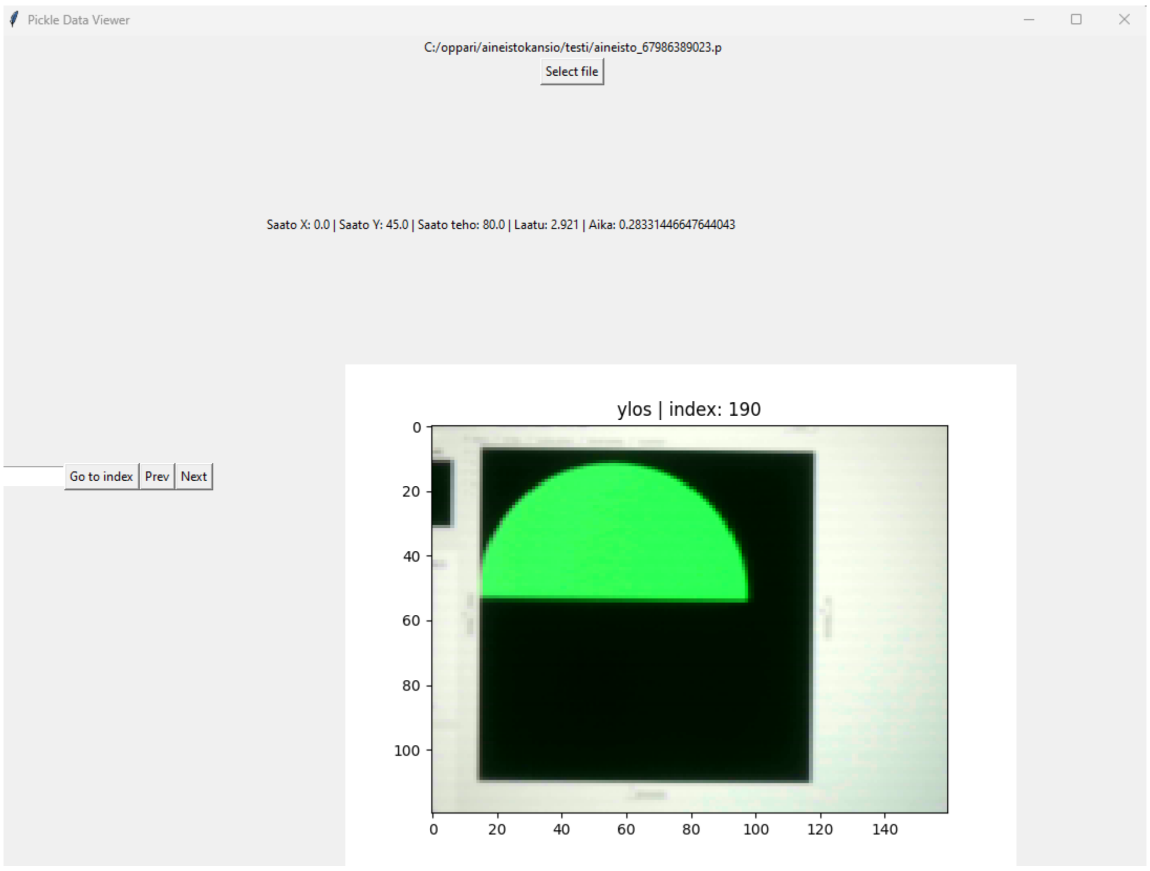
The process of collecting images in pickle files is rather time-consuming, especially when using the laptop that was employed. With approximately 30 GB of learning and testing data (comprising about 2800 files), it took approximately two days and twelve hours. This is also the reason why the data was divided into smaller files. Additionally, storing the data in RAM memory and saving large files resulted in the computer slowing down and potentially crashing the entire system.

3.3.2 Utility Program Pickle Data Viewer for Validate Collected Data

The Collected training data are stored in Pickle files, which are binary files. To validate the data inside Pickle file, it is necessary to open them. To this end, a simple application has been developed, which allow the user to easily open files to see visualized images and validate the data from files manually when needed. For this purpose, a simple utility program called Pickle Data Viewer has been developed to display images and associated data stored in those files. This program is specifically designed to work with data stored by the application used in the data collection process.

Having open files is imperative for validating the alignment of images and labels and identifying potential data corruption. Any corrupted files must be identified and removed. The same application can also be employed to check images that have been incorrectly predicted during the testing phase after the Convolutional Neural Network (CNN) model has been created. This aids in understanding the reasons behind prediction errors.

The complete Code for this Picke Data Viewer application can be found in Appendix 1.



Saato X: 0.0 | Saato Y: 45.0 | Saato teho: 80.0 | Laatu: 2.921 | Aika: 0.28331446647644043

FIGURE 6. Picture of Pickle Data Viewer UI with one opened image

Figure 6 illustrates an example of an opened pickle file, the file named *'aineisto_67986389023.p'* displaying the image at index 190. The image labels indicate that the hemisphere shape is 'ylos' (translated to 'up' in English), which is consistent with what is observable in the image.

3.4 Training Convolutional Neural Network / Implementation

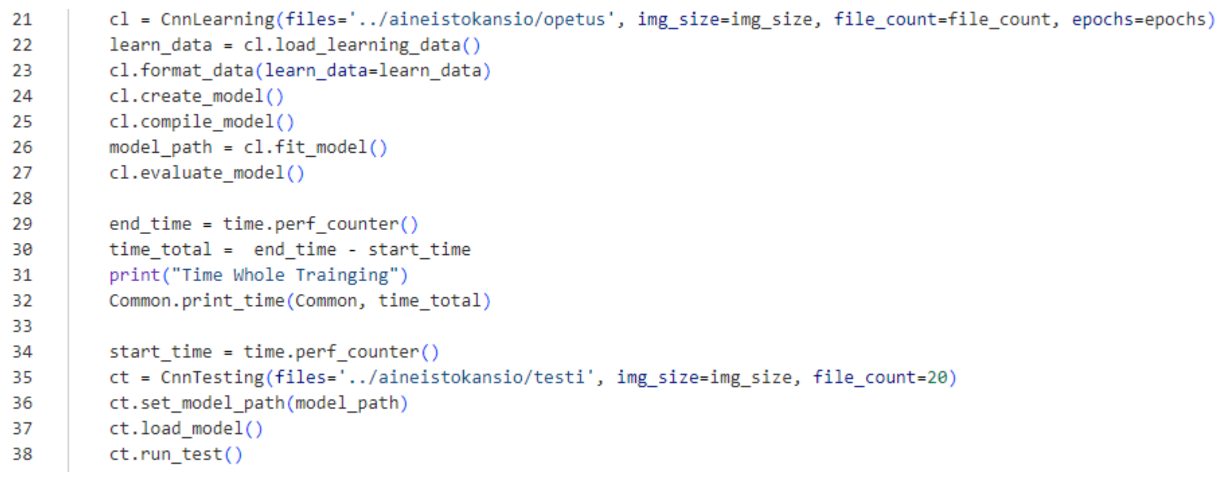
The training code for the images was composed in the Python programming language. The codes are segregated into four distinct Python classes. One class (Combined) serves as the main category, with an actual training class (CnnLearning), a testing class (CnnTesting), and a common class (Common) encompassing codes shared between training and testing. The training and testing images are stored in separate folders.

3.4.1 Combined Class

Within the Combined class, parameters can be easily provided for the CnnLearning class, specifying the desired image size, the number of training files to be utilized, the desired number of epochs for training the Convolutional Neural Network (CNN) model, and the folder where the training images are located. Adjusting these parameters allows for the examination of how effectively a processing machine can perform a given task.

Similarly, the CnnTesting class can be supplied with parameters, including image size, the number of test files for evaluation, and the folder where the test images are stored.

The following code is a snippet from the *'combined.py'* code. The complete code can be found in Appendix 3.



```
21        cl = CnnLearning(files='../aineistokansio/opetus', img_size=img_size, file_count=file_count, epochs=epochs)
22        learn_data = cl.load_learning_data()
23        cl.format_data(learn_data=learn_data)
24        cl.create_model()
25        cl.compile_model()
26        model_path = cl.fit_model()
27        cl.evaluate_model()
28
29        end_time = time.perf_counter()
30        time_total =  end_time - start_time
31        print("Time Whole Trainging")
32        Common.print_time(Common, time_total)
33
34        start_time = time.perf_counter()
35        ct = CnnTesting(files='../aineistokansio/testi', img_size=img_size, file_count=20)
36        ct.set_model_path(model_path)
37        ct.load_model()
38        ct.run_test()
```

FIGURE 7. Code snippet from combined.py code

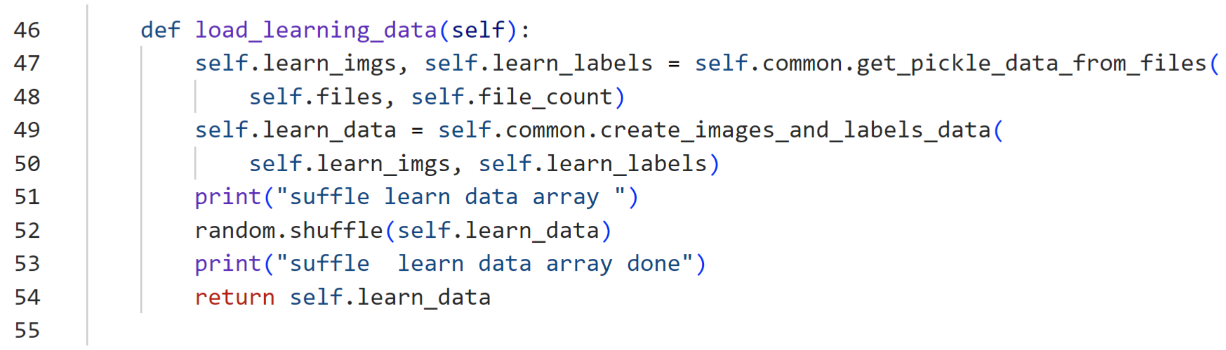
From lines 21 to 27, a CnnLearning object is instantiated, and its methods are utilized to preprocess data for machine learning algorithms, create, compile, and fit the model.

From lines 35 to 38, the previously created model is utilized for testing images to evaluate its performance.

3.4.2 CnnLearning Class

In the CnnLearning class, the data has methods for loading data, formatting data, compiling the model, fitting the model, and evaluating the model. The learning process follows a sequential order. Initially, data needs to be loaded and formatted to be suitable for training the model. Subsequently, the model is created, which involves adding convolution, activation, and pooling layers with filters. Once the model is created, it is compiled first and then fitted. The fitting process involves using TensorFlow library method to train the Convolutional Neural Network (CNN) model. After training the data, the model's accuracy in predicting given images can be validated using the evaluate method.
The complete code is available in Appendix 4.

The initial step involves loading the learning data.



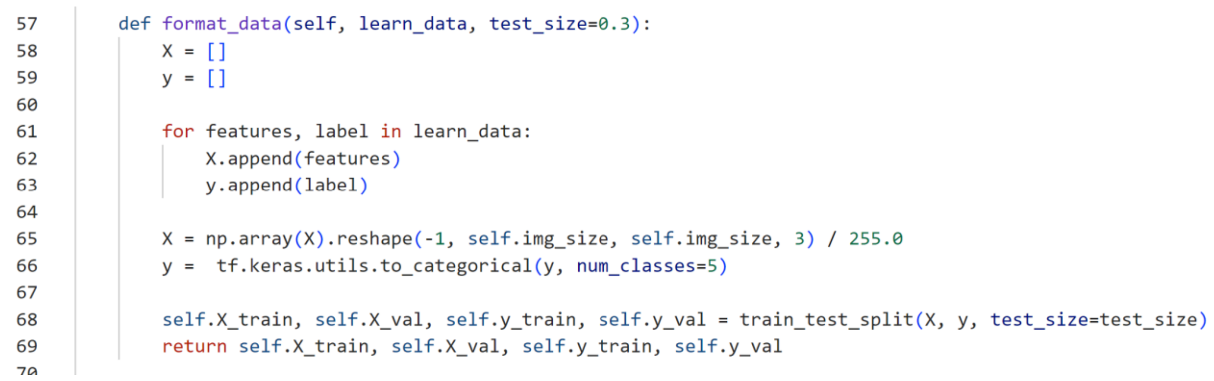
```
46      def load_learning_data(self):
47          self.learn_imgs, self.learn_labels = self.common.get_pickle_data_from_files(
48              self.files, self.file_count)
49          self.learn_data = self.common.create_images_and_labels_data(
50              self.learn_imgs, self.learn_labels)
51          print("suffle learn data array ")
52          random.shuffle(self.learn_data)
53          print("suffle  learn data array done")
54          return self.learn_data
55
```

FIGURE 8. Code snippet from CnnLearning.py code

On line 47, a method from the Common class is employed to retrieve data from a pickle file. This retrieval results in arrays of images and their corresponding labels.

On line 49, a method from the Common class, `*create_images_and_labels_data*`, is utilized. This method returns new arrays of data. Each array is populated with elements containing an actual image and its corresponding label. The image is parsed and resized from binary using the resize method, and the label is converted from a string to an integer format. The resulting array, `*learn_data*`, is then returned.

The second step is to format the data in a suitable format for training a Convolutional Neural Network.



```
57      def format_data(self, learn_data, test_size=0.3):
58          X = []
59          y = []
60
61          for features, label in learn_data:
62              X.append(features)
63              y.append(label)
64
65          X = np.array(X).reshape(-1, self.img_size, self.img_size, 3) / 255.0
66          y =  tf.keras.utils.to_categorical(y, num_classes=5)
67
68          self.X_train, self.X_val, self.y_train, self.y_val = train_test_split(X, y, test_size=test_size)
69          return self.X_train, self.X_val, self.y_train, self.y_val
70
```

FIGURE 9. Code snippet from CnnLearning.py code

From lines 61 to 63, the `*learn_data*` variable is divided into two arrays, X and y. In the X array, images are appended as features, while the y array, labels are appended.

On line 65, array X is reshaped into a numpy array. The parameters provided to the reshape method are as follows: -1 automatically determines the array size based on the number of elements in the X array. The image sizes represent the width and height dimensions targeted for reshaping. The last parameter specifies the number of image channels; 3 is used for RGB images. Then all elements in the X array are normalized by dividing each element by 255. This process scales all pixel values to the range of 0 to 1, which facilitates the training process.

On line 66, the array y, which contains five different categories, is transformed into an array matrix with binary values (0 or 1). The resulting matrix has columns that correspond to the number of categories present in the data.

On line 68, the arrays X and y are divided into two distinct data arrays, one designated for training and the other for testing. These arrays are then returned to the function caller on the next line.

3.4.3 CnnTesting Class

The CnnTesting class was employed for the purpose self-validation testing for the generated model. This was not within the scope of the study, but it was included here because it was used as a validation. This class randomly retrieves images from the test folder. It runs them through the generated model and determines how well the generated model was able to predict the shape of the sphere. The complete code for this can be found in Appendix 5.

3.4.4 Common Class

The Common class is employed for all common methods utilized by other python classes. The complete code can be found in Appendix 6.

3.5 Training with Supercomputer

The utilization of the supercomputer Puhti requires some adaptations compared to a regular computer because it needs to be accessed over the internet, and initiating tasks operates differently. The procedures required to execute the convolutional neural network training software will be elucidated next with Puhti.

The training and testing data images were transferred to the Allas service, which is CSC's object storage system. This object storage system is designed for storing large amounts of data, as disk space and the number of files is limited on the Puhti machine.

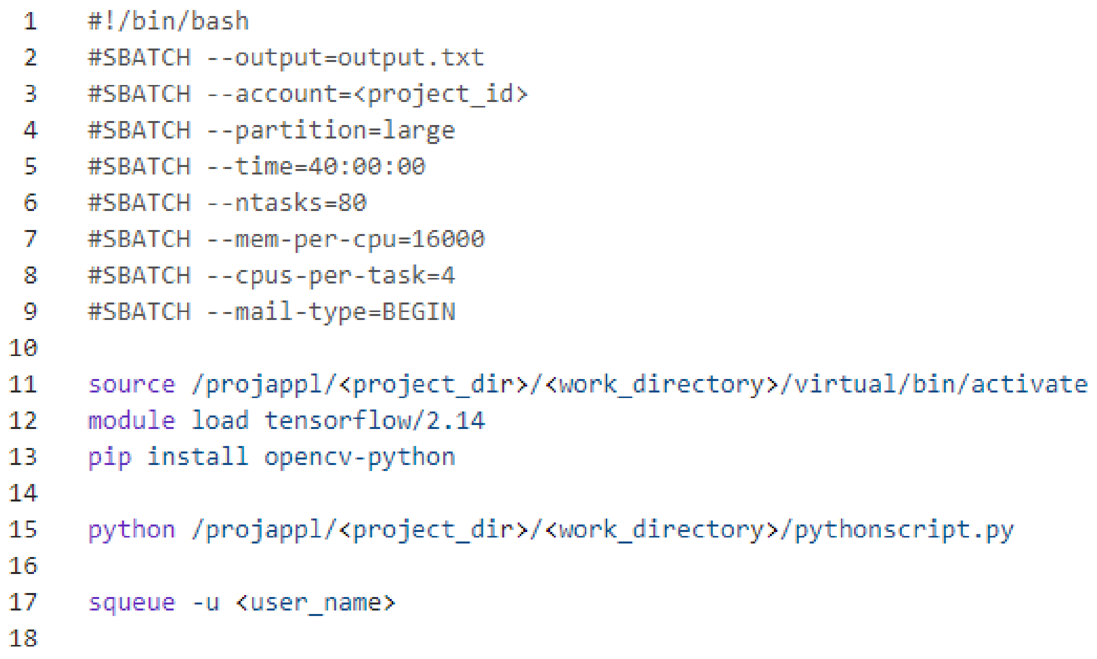
The transfer of materials was done conducted via the *`SCP`* (Secure Copy Protocol) program, as the mount of training data was over 30Gb. This is a secure and reliable format for transferring data.

For instance, by executing the command:
‘scp -r /local/directory/ username@puhti.csc.fi:/scratch/<project_directory>/’
This command copies the directory from the local host to a remote host using SCP.

The executable codes were transferred via GitHub. Their modification was performed directly on the Puhti machine using SSH over the terminal. The connection is established using the command *'ssh <username>@puhti.csc.fi'*.

The Puhti machine is utilized to run the codes through the Slurm batch job system. Jobs are not executed immediately but are queued for processing. Various parameters, such as runtime, memory allocation, and the number of cores, must be defined for the execution. The configuration is done through a Bash script.

The following code snippet is an of example of a job script that was used in the thesis.



```
 1  #!/bin/bash
 2  #SBATCH --output=output.txt
 3  #SBATCH --account=<project_id>
 4  #SBATCH --partition=large
 5  #SBATCH --time=40:00:00
 6  #SBATCH --ntasks=80
 7  #SBATCH --mem-per-cpu=16000
 8  #SBATCH --cpus-per-task=4
 9  #SBATCH --mail-type=BEGIN
10
11  source /projappl/<project_dir>/<work_directory>/virtual/bin/activate
12  module load tensorflow/2.14
13  pip install opencv-python
14
15  python /projappl/<project_dir>/<work_directory>/pythonscript.py
16
17  squeue -u <user_name>
18
```

FIGURE 10. Example of Puhti computer sbatch job script

From lines 2 to 9, specify settings for the Slurm system, which control and manage the execution of tasks on computing clusters. The settings cover parameters such as output file name, project account, the chosen partition, the number of tasks to be executed, the number of CPU cores, memory requirements, the maximum allowed runtime, and a request for GPU resources.

The variables that were adjusted for different executions were time, ntasks, mem-per-cpu, and cpu-per-task. In particularly, the runtime and memory had to be increased when the number of epochs and training images was increased.

From line 11 to 15, The script activates a virtual environment, loads the TensorFlow module, installs the OpenCV Python package, and finally run a Python script 'compinend.py'.

To submit a job on Puhti, execute the command `*sbatch <script_name>.sh*`. Following submission, monitor the queue status using `*squeue -u $<username>*`.

3.6 GFLOPS calculation

To calculate the comparison value between the computing power of different computers, a straight-forward Python program was developed. The program reads the machine's processor data, per-forms a matrix multiplication on a 10000 x 10000 random number matrix, and uses the elapsed time to calculate the GFLOPS of the machine performing the operation. The code was tested against the Linx (LinPack) tool (https://soft4bro.com/soft/linx-linpack-download) and the results were nearly the same on the laptop. The complete code for flops.py can be found in Appendix 2.

4 RESULTS

The data collected in the thesis is being examined to answer the question of when it is advisable to switch to using a supercomputer instead of a regular computer.

For the research, the training algorithm was run through various combinations. The execution times, obtained accuracies, and loss values were recorded from the runs, as well as how well the predictions succeeded with our custom test software. The execution times were recorded for the entire software performance, including both the loading of the training data and the duration of the training phase itself. The duration of the execution was also captured from the moment the program was queued on the supercomputer. On the supercomputer, it is necessary to perform specific installations for running the software, so this is also essential information.

The computational performance of the machines was tested using a dedicated GFLOPS calculator, from which the computational capabilities of the machine were recorded.

Results were collected by varying the size of the image used for training, thereby altering the number of parameters in the model. The image sizes used were 60x60, 70x70, 80x80, and 100x100 pixels in width and height. Additionally, the quantities of pickle files used for training were varied, each containing approximately two hundred training images. The file quantities used were 50, 100, 500, and 750 for the local machine, and for the Puhti supercomputer, additional tests were conducted with 1000, 1500, and 2000 files. The capacity of the regular machine reached its limit at 750, but larger quantities could be allocated to the supercomputer.

The results will be reviewed initially for each machine, followed by an examination comparing them with each other.

4.1 FLOPS

Flops test was done with 10000 x 10000 random matrix. Larger matrix will overload the laptop and was therefore ignored.

TABLE 1. Calculate GFLOPS by multiplicate 10000 x 10000 random matrix

Computer	**GFLOPS**	**Calculation time [s]**
Laptop	70	28
Supercomputer	630	3

Table 1 presents the outcomes of the GFLOP calculation for both the laptop and supercomputer. GFLOPs indicate the result obtained, while calculation time indicates the time taken to complete the counter. Of these, the most pertinent value for investigation is GFLOPS, which indicates that, according to this, the supercomputer is approximately 9 times more efficient. The difference between the observed and theoretical (17000 x) results is not as pronounced as it could be, as the focus of this work was not on optimizing the running environments and codes. The runs were performed with settings that were relatively basic.

4.2 Convolutional Neural Network Training

TABLE 2. Image size 70x70 pixels, epochs 10, batch size 32, kernel size 3, base filter size 32. Times in seconds

Pickle file count	**Image count**	**Images size (Gb)**	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**
50	10050	0.55	310	316	29	36
100	20100	1.1	563	574	57	69
500	100500	5.4	4666	5092	283	341
750	150710	8.1	21179	23268	424	500
1000	210000	10.8	N/A	N/A	567	651

Table 2 presents the results of running the Convolutional Neural Network (CNN) training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 70x70 pixels. In the training, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training. In the case of a pickle count of 1000, the times for the laptop are not applicable, as the laptop memory is incapable of loading the size of the images.

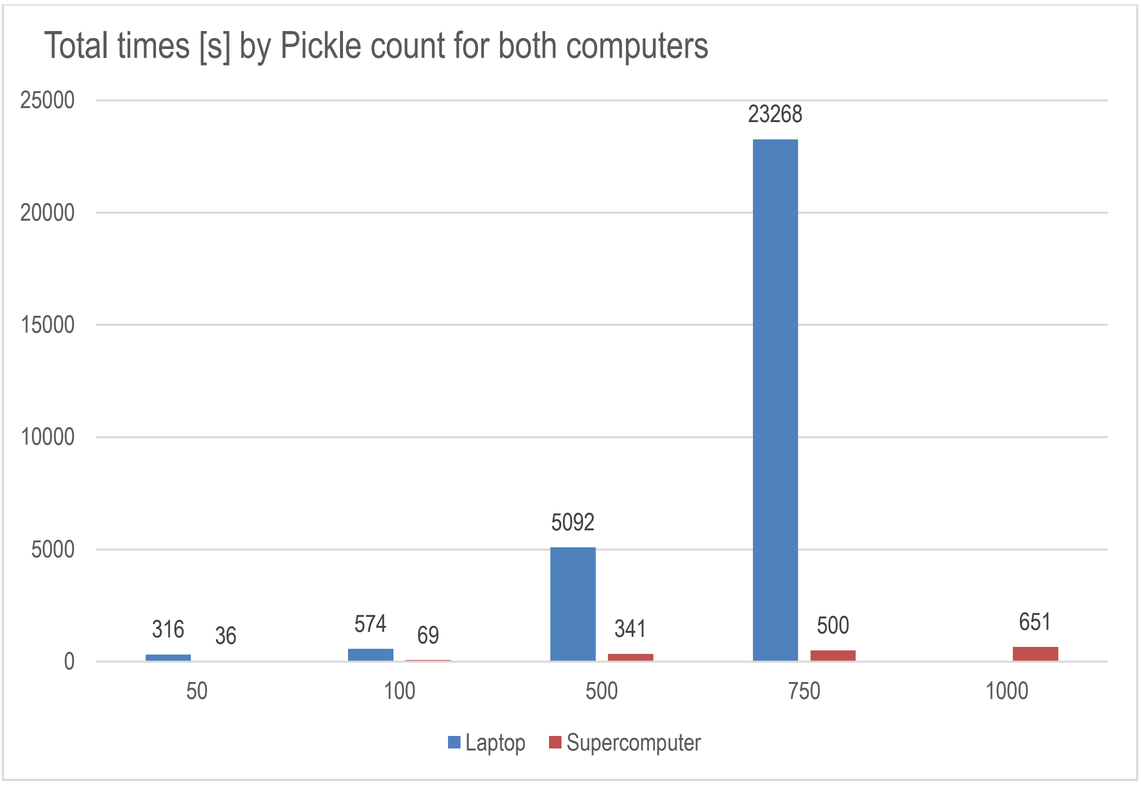


FIGURE 11. Total times in seconds by Pickle count for image size 70x70

Figure 11 presents a bar chart of the data in Table 2, which compares only the total time taken to run the program. The figure clearly illustrates how the execution time on the laptop begins to increase rapidly as the supercomputer steadily completes the task.

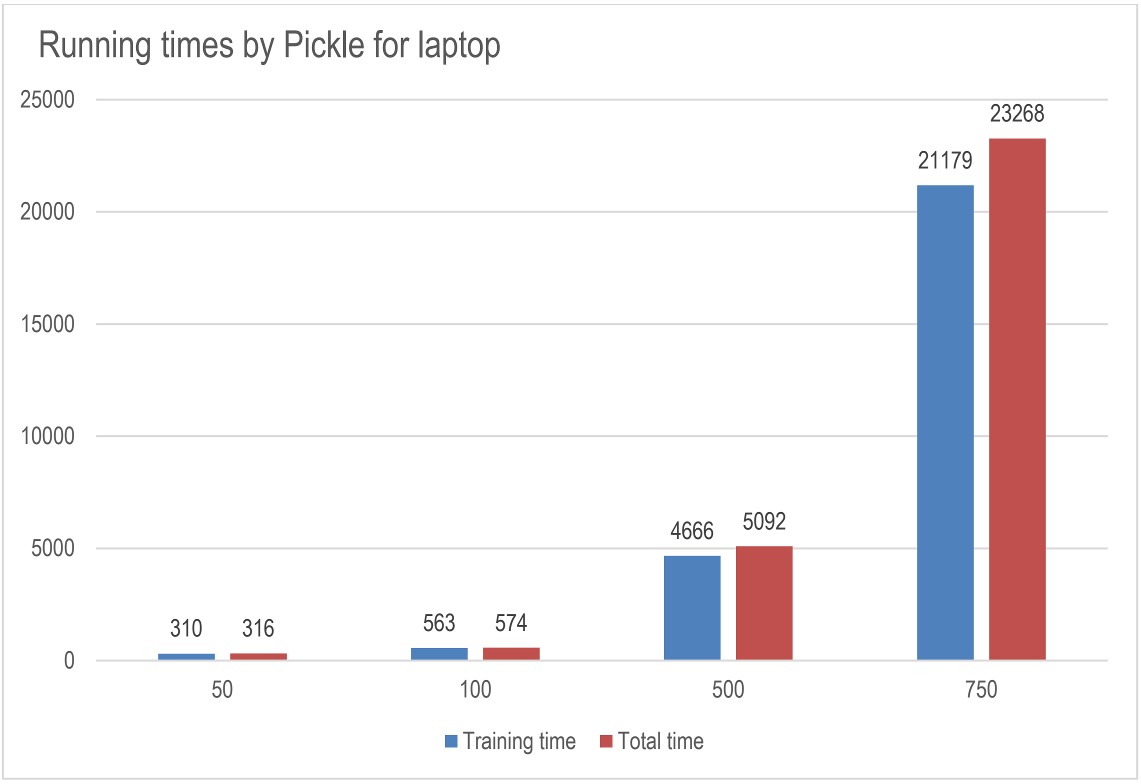


FIGURE 12. Running time in seconds by Pickle for image size 70x70 with laptop

Figure 12 presents a bar chart of the data in Table 2, which compares only the total time spent on the laptop and the time spent on teaching. The figure clearly demonstrates that most of the execution time is spent on CNN training.

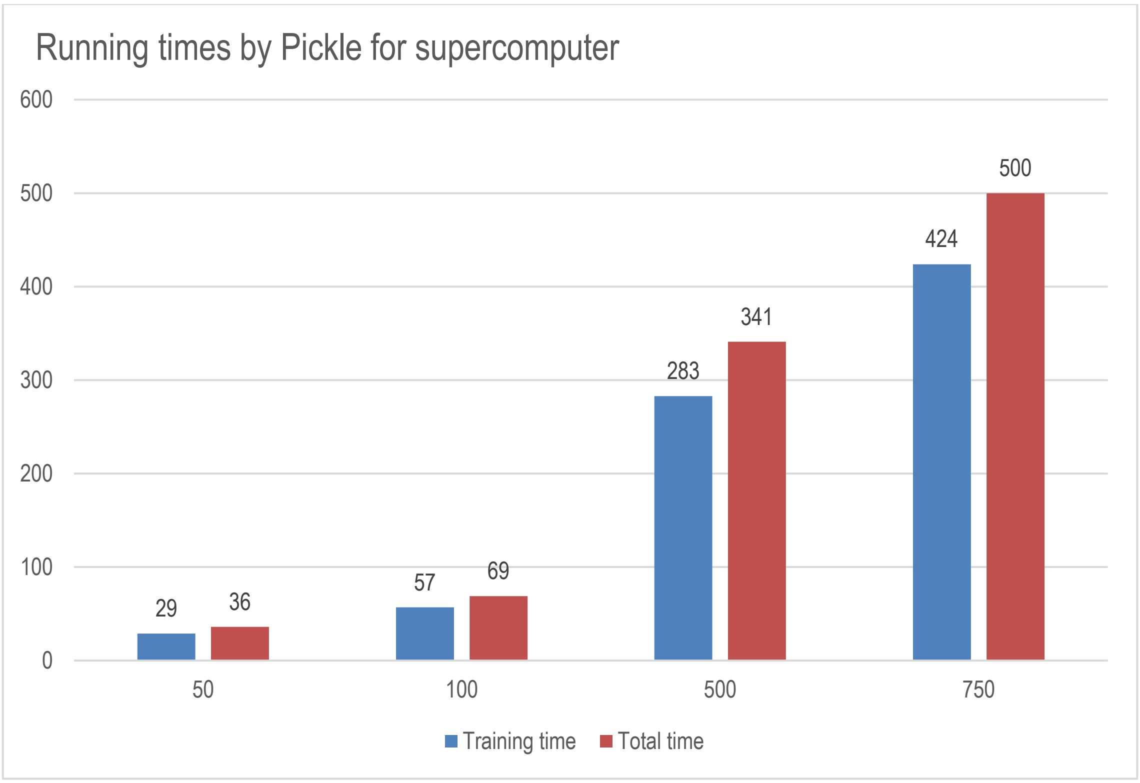


FIGURE 13. Running time in seconds by Pickle for image size 70x70 with supercomputer

Figure 13 presents a bar chart of the data in Table 2, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 3. Image size 60x60 pixels, epochs 10, batch size 32, kernel size 3, base filter size 32

Pickle file count	**Image count**	**Images size (Gb)**	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**
50	10050	0.55	231	237	24	41
100	20100	1.1	552	566	43	67
500	100500	5.4	5514	5769	213	356
750	150710	8.1	15444	17271	321	492
1000	200960	10.8	24308	27540	424	588

Table 3 presents the results of running the CNN training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 60x60 pixels. In the tutorial, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training.

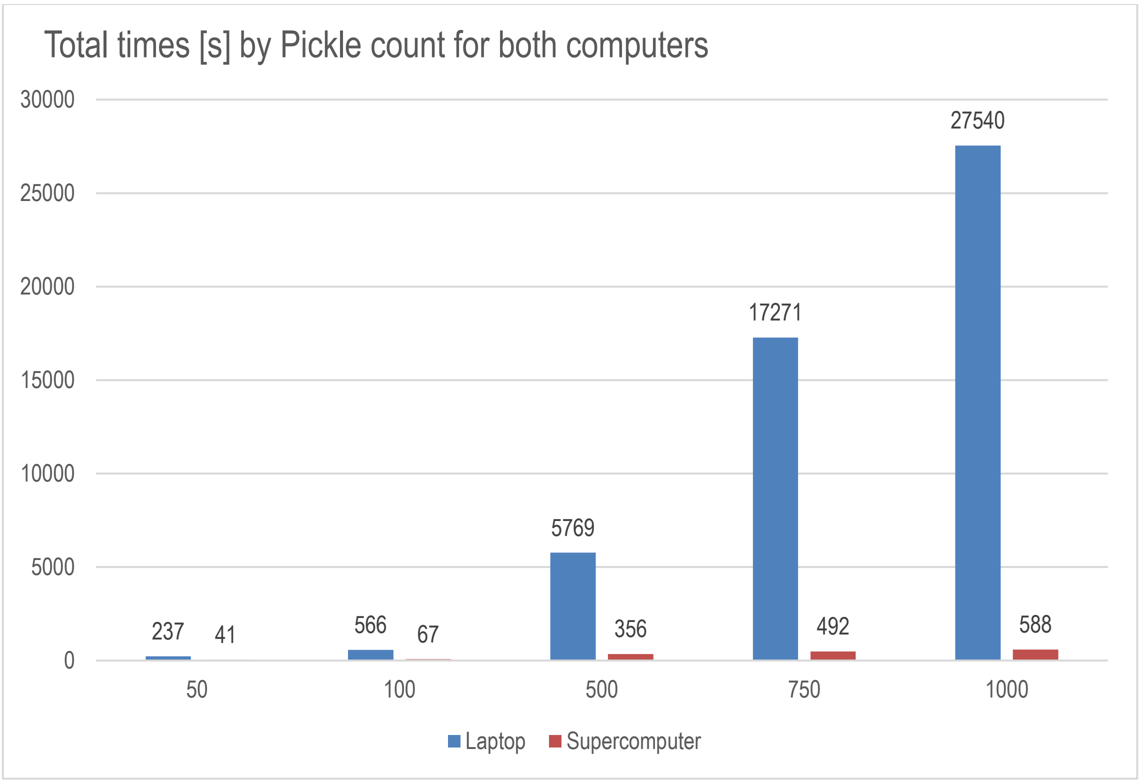


FIGURE 14. Total times in seconds by Pickle count for image size 60x60

Figure 14 presents a bar chart of the data in Table 3, which compares only the total time taken to run the program. The figure clearly illustrates how the execution time on the laptop begins to increase rapidly while the supercomputer increases much moderately.

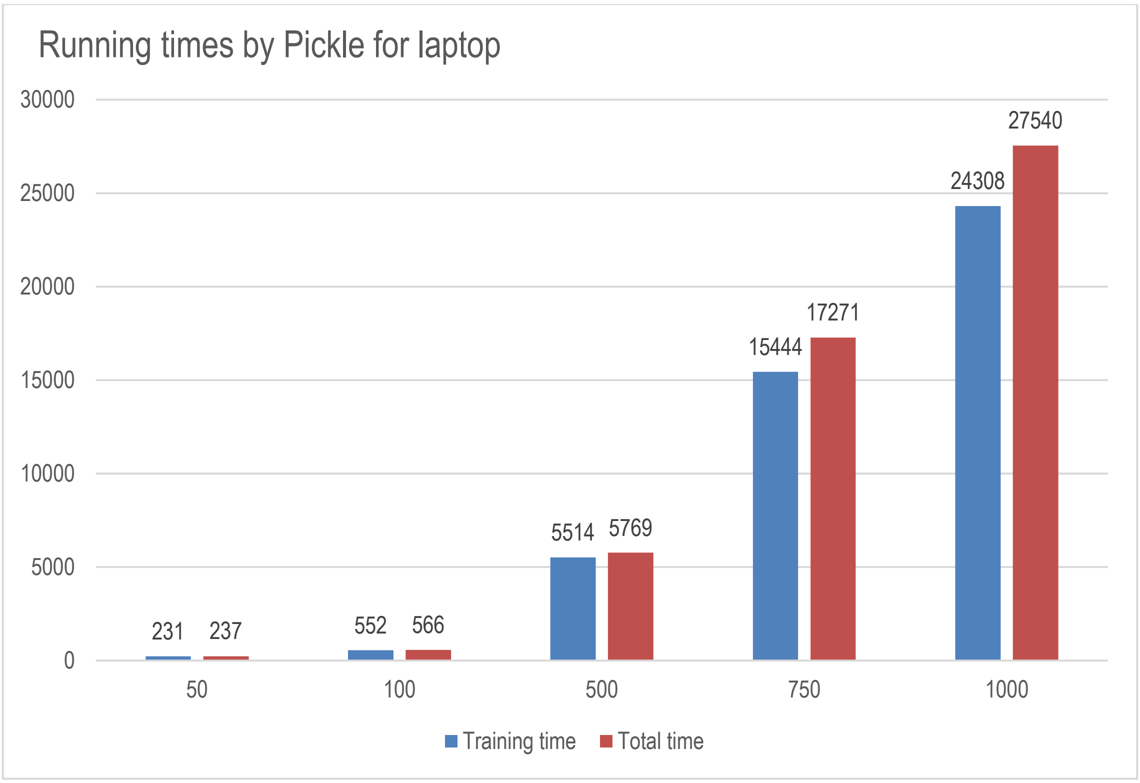


FIGURE 15. Running time in seconds by Pickle for image size 60x60 with laptop

Figure 15 presents a bar chart of the data in Table 3, which compares only the total time spent on the laptop and the time spent on teaching. The figure clearly demonstrates that most of the execution time is spent on CNN training.

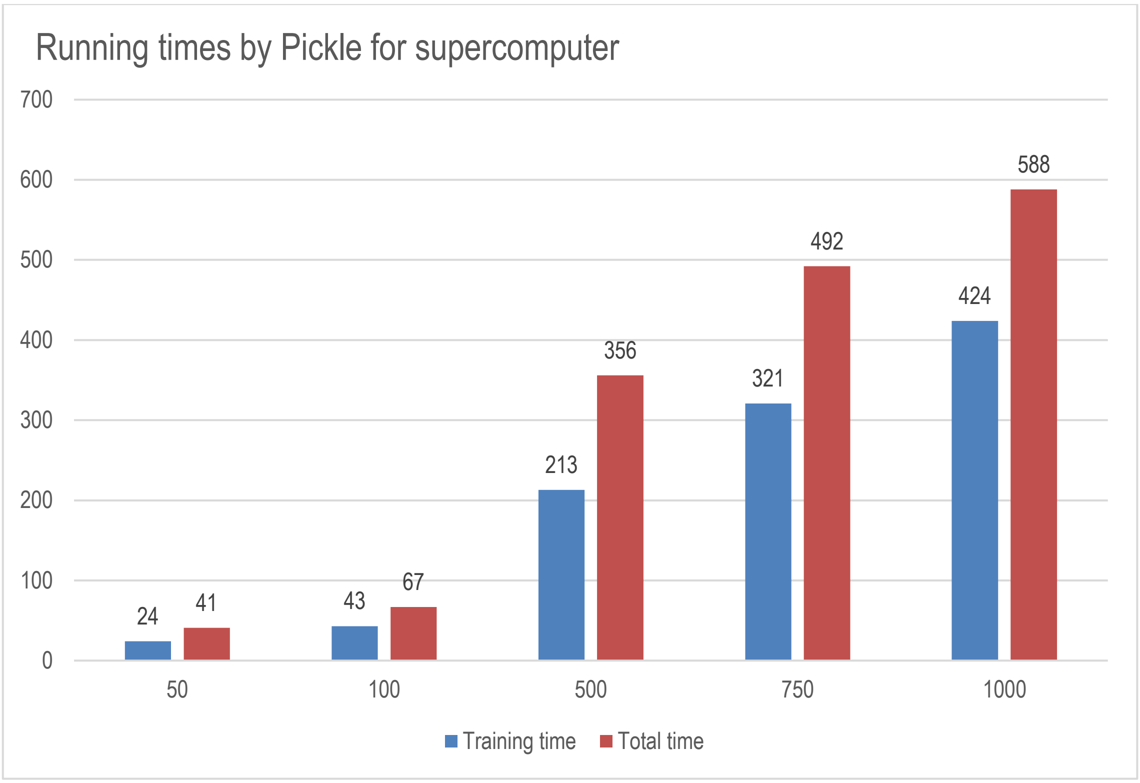


FIGURE 16. Running time in seconds by Pickle for image size 60x60 with supercomputer

Figure 16 presents a bar chart of the data in Table 3, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 4. Image size 80x80 pixels, epochs 10, batch size 32, kernel size 3, base filter size 32

Pickle file count	**Image count**	**Images size (Gb)**	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**
50	10050	0.55	776	794	35	45
100	20100	1.1	2160	2221	76	91
500	100500	5.4	10612	11609	351	432
750	150710	8.1	N/A	N/A	524	649

Table 4 presents the results of running the CNN training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 80x80 pixels. In the training, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training. In the case of a pickle count of 750, the times for the laptop are not applicable, as the laptop memory is incapable of loading the size of the images.

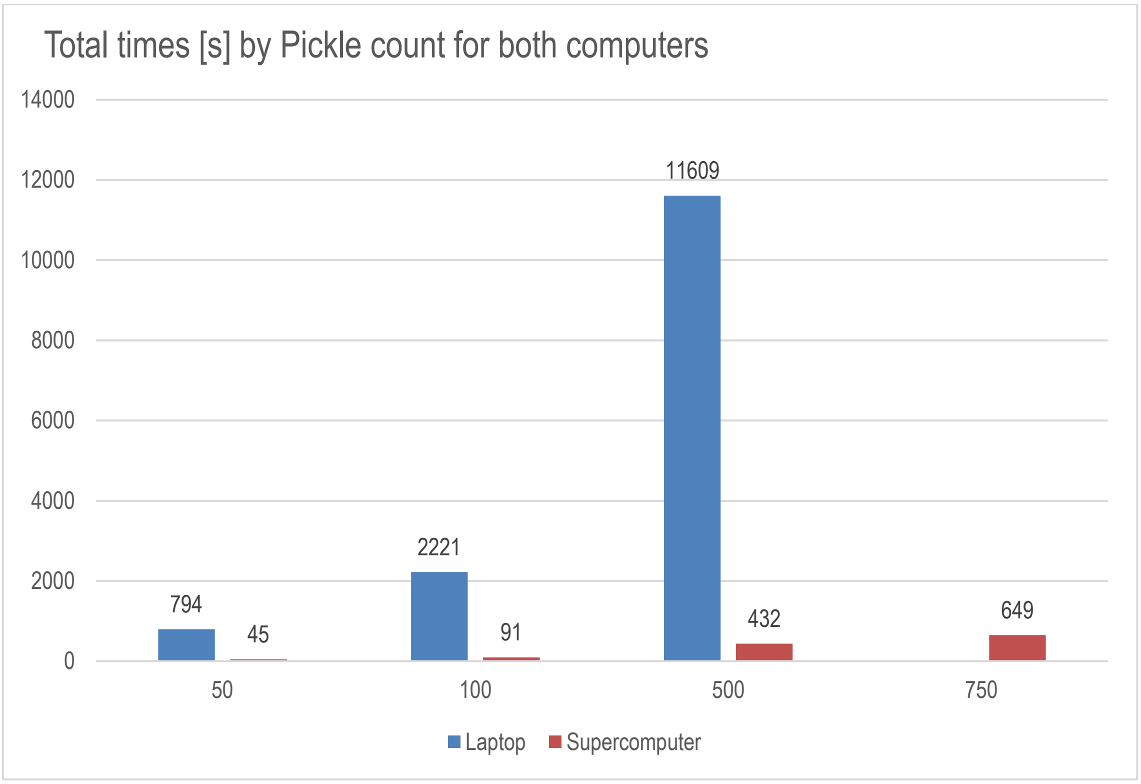


FIGURE 17. Total times in seconds by Pickle count for image size 80x80

Figure 17 presents a bar chart of the data in Table 4, which compares only the total time taken to run the program. The figure clearly illustrates how the execution time on the laptop begins to increase rapidly while the supercomputer increases much moderately. The laptop result for 750 is missing because the laptop memory is unable to accommodate the size of the data.

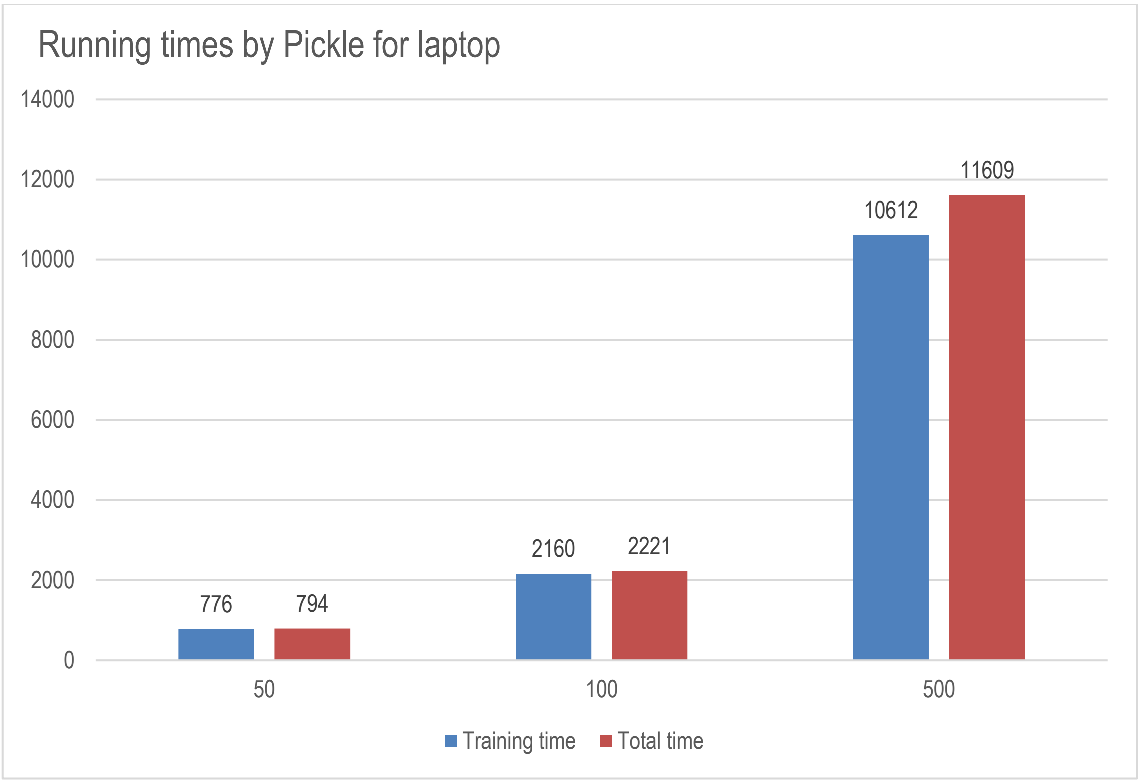


FIGURE 18. Running time in seconds by Pickle for image size 80x80 with laptop

Figure 18 presents a bar chart of the data in Table 4, which compares only the total time spent on the laptop and the time spent on teaching. The figure clearly demonstrates that most of the execution time is spent on CNN training.

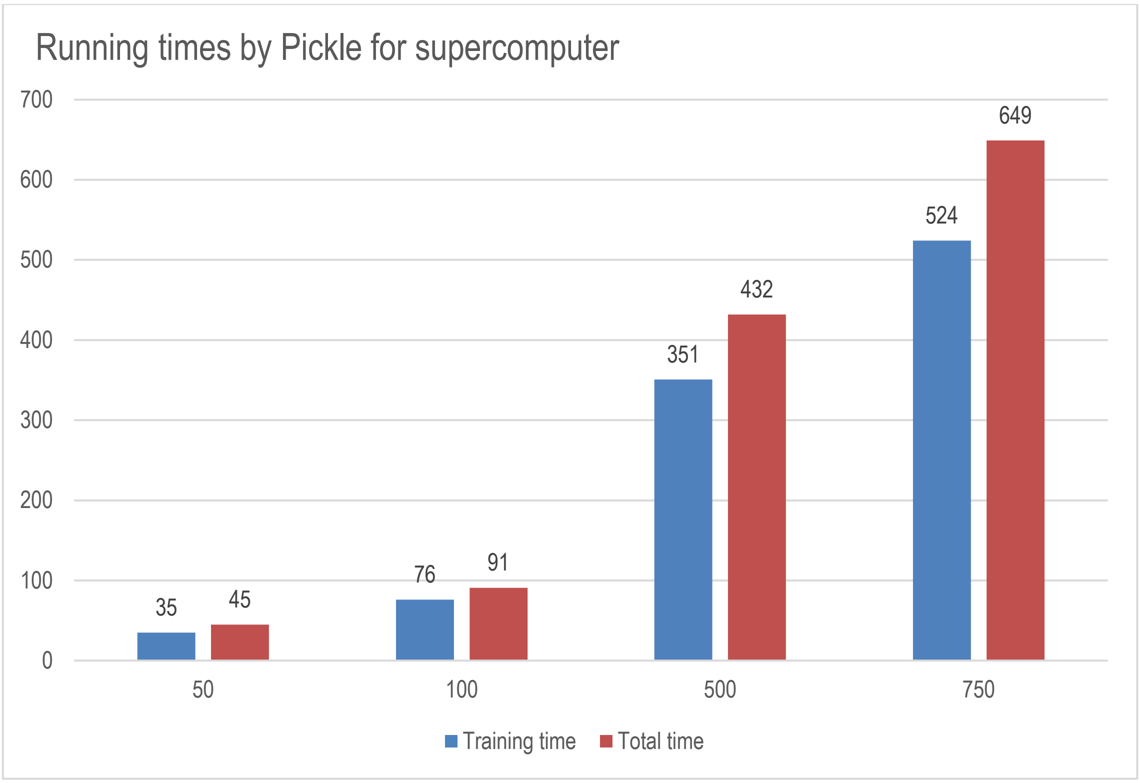


FIGURE 19. Running time in seconds by Pickle for image size 80x80 with supercomputer

Figure 19 presents a bar chart of the data in Table 4, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 5. Image size 100x100 pixels, epochs 10, batch size 32, kernel size 3, base filter size 32

Pickle file count	Image count	Images size (Gb)	Laptop training time [s]	Laptop total time [s]	Supercomputer training time [s]	Supercomputer total time [s]
50	10050	0.55	1107	1125	52	75
100	20100	1.1	2381	2478	105	137
500	100500	5.4	13905	14529	522	641
1000	201000	10.8	N/A	N/A	1047	1307
1500	301500	16.2	N/A	N/A	1577	1901
2000	401960	21.6	N/A	N/A	2100	2543

Table 5 presents the results of running the CNN training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 100x100 pixels. In the training, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training. In the case of a pickle count of 1000 and above, the times for the laptop are not applicable, as the laptop memory is incapable of loading the size of the images.

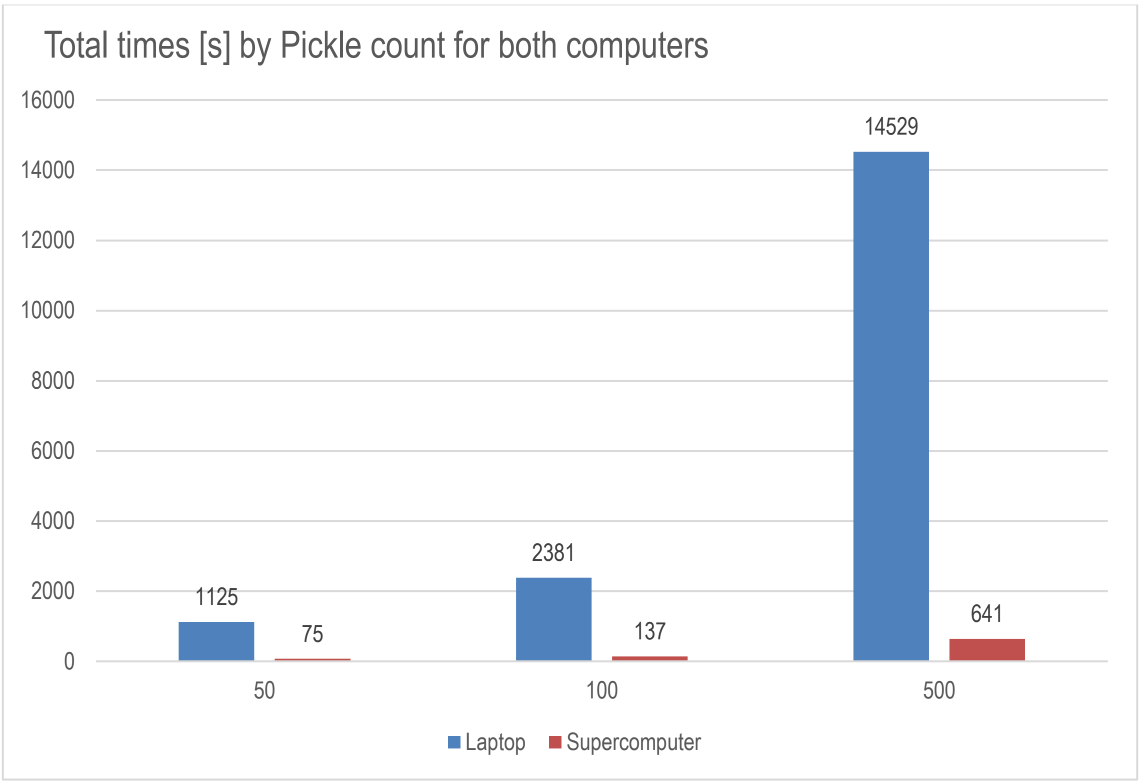


FIGURE 20. Total times in seconds by Pickle count for image size 100x100

Figure 20 presents a bar chart of the data in Table 5, which compares only the total time taken to run the program. The figure clearly illustrates how the execution time on the laptop begins to increase rapidly while the supercomputer increases much moderately.

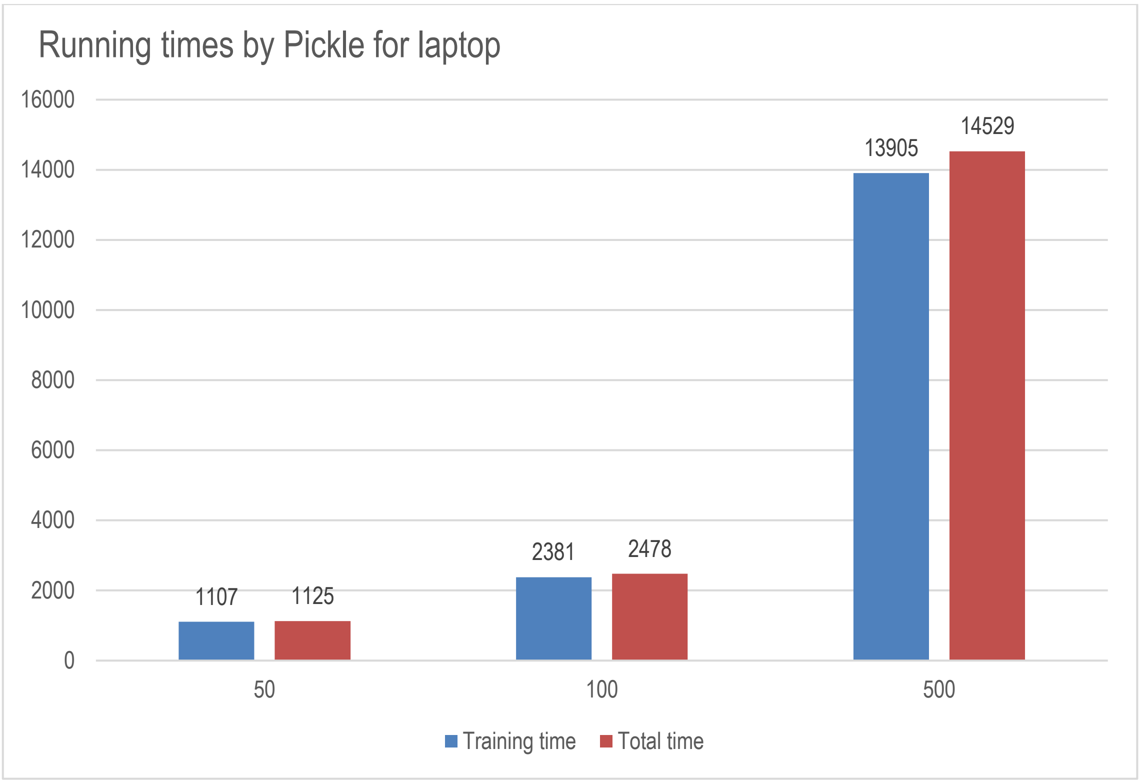


FIGURE 21. Running time in seconds by Pickle for image size 100x100 with laptop

Figure 21 presents a bar chart of the data in Table 5, which compares only the total time spent on the laptop and the time spent on teaching. The figure clearly demonstrates that most of the execution time is spent on CNN training.

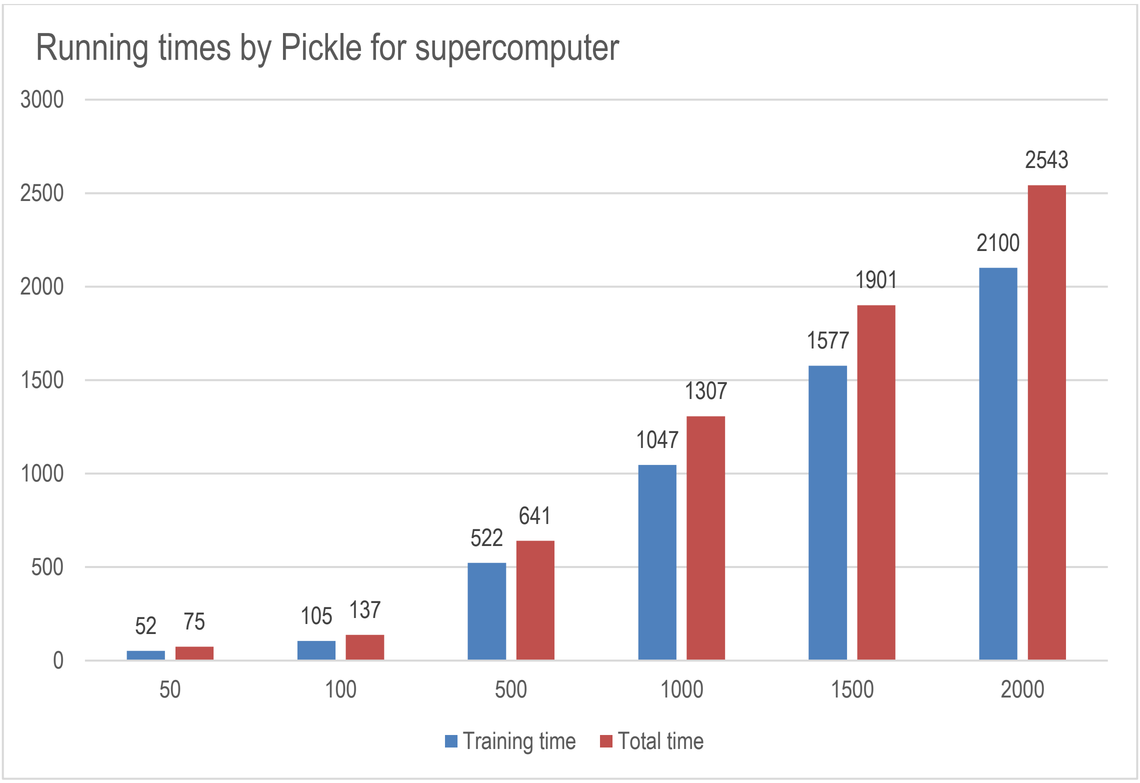


FIGURE 22. Running time in seconds by Pickle for image size 100x100 with supercomputer

Figure 22 presents a bar chart of the data in Table 5, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 6. Image size 70x70 pixels, epochs 10, batch size 64, kernel size 5, base filter size 64

Pickle file count	Image count	Images size (Gb)	Laptop training time [s]	Laptop total time [s]	Supercom-puter training time [s]	Supercom-puter total time [s]
100	20100	1.1	4113	4376	144	153
500	100500	5.4	9327	9647	742	830
750	150710	8.1	13826	14335	1109	1227

Table 6 presents the results of running the CNN training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 70x70 pixels. In the training, 10 epochs are used, the kernel size is set to 5, and the batch size and base filter are set to 64. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training.

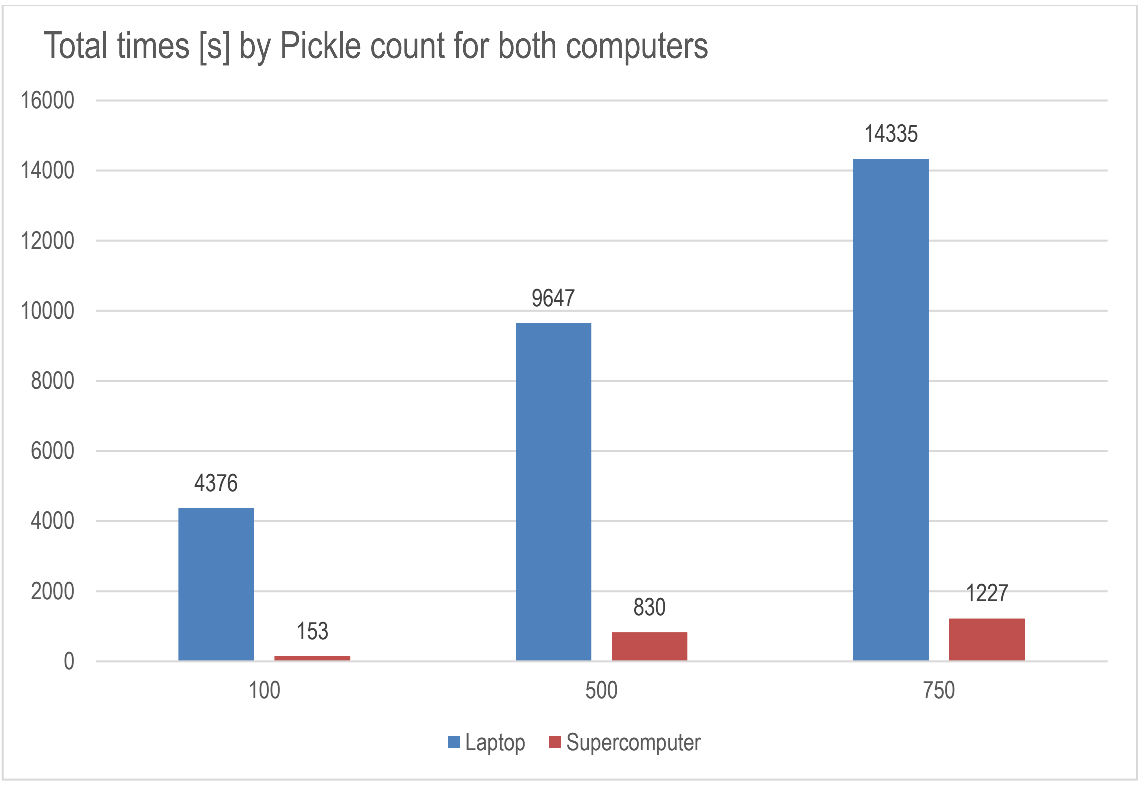


FIGURE 23. Total times in seconds by Pickle count for image size 70x70, batch size 64, kernel size 5 and base filter size 64

Figure 23 presents a bar chart of the data in Table 6, which compares only the total time taken to run the program. The figure clearly illustrates how the execution time on the laptop begins to increase rapidly while the supercomputer increases much moderately.

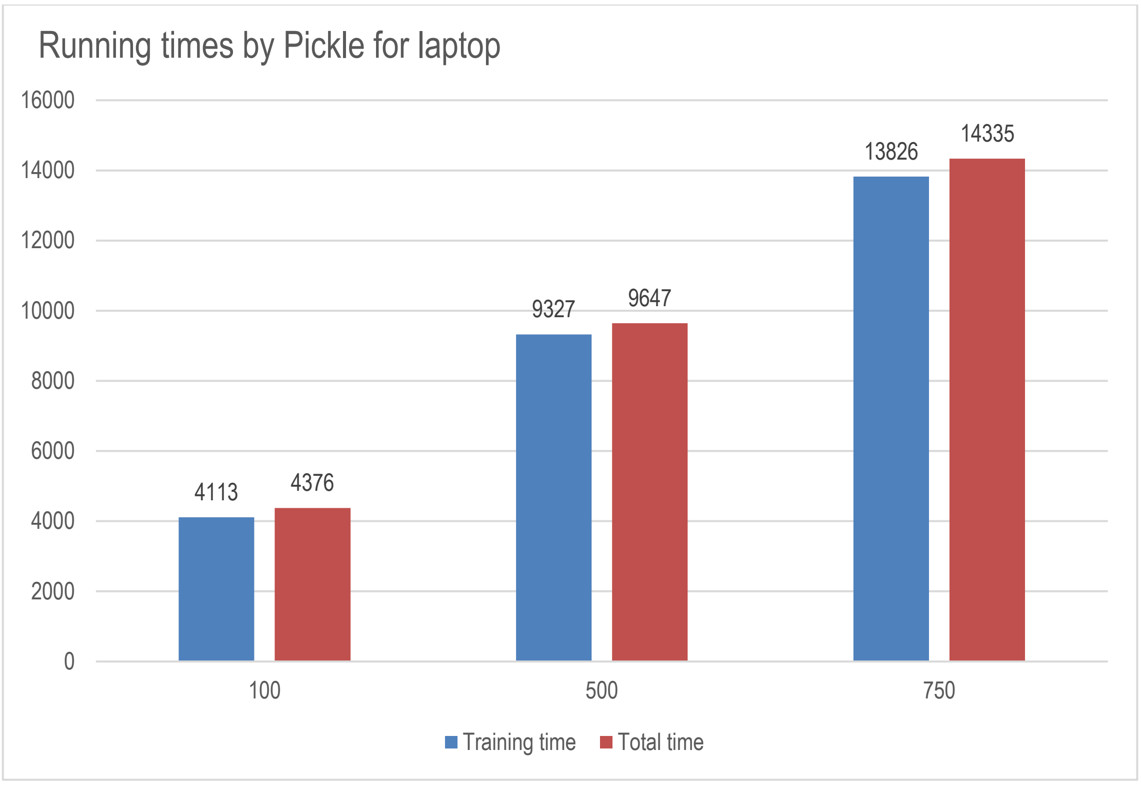


FIGURE 24. Running time in seconds by Pickle for image size 70x70, batch size 64, kernel size 5 and base filter size 64 with laptop

Figure 24 presents a bar chart of the data in Table 6, which compares only the total time spent on the laptop and the time spent on teaching. The figure clearly demonstrates that most of the execution time is spent on CNN training.

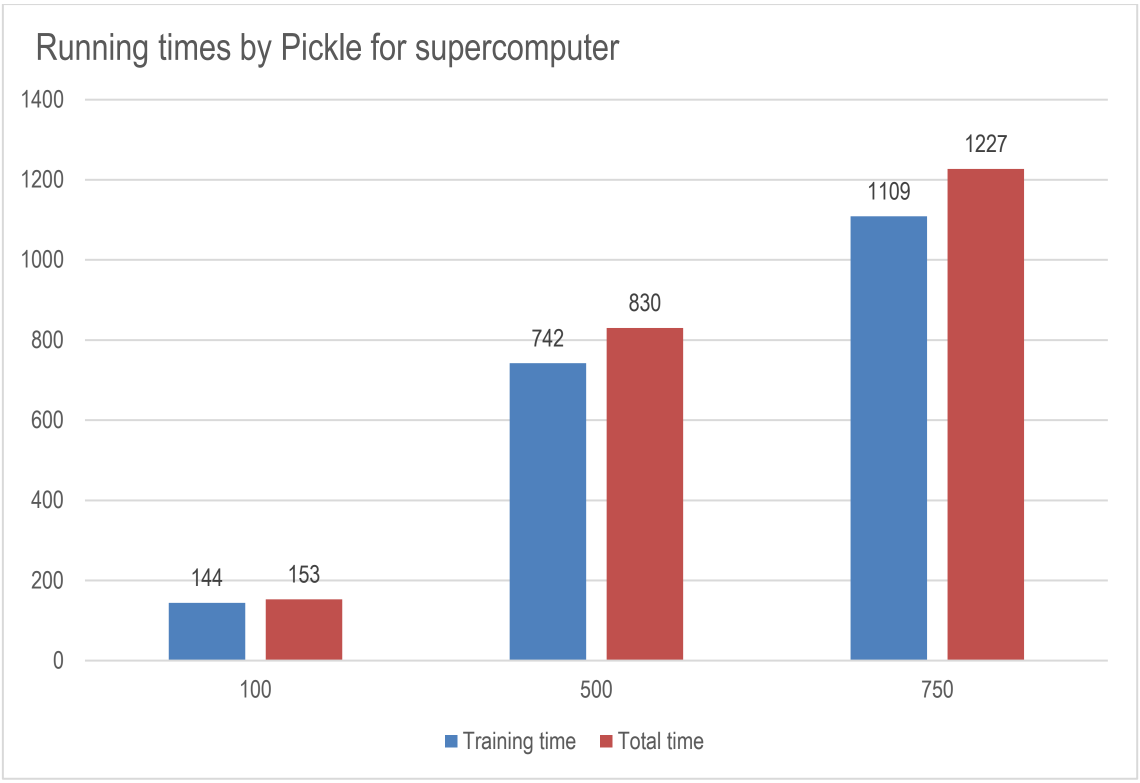


FIGURE 25. Running time in seconds by Pickle for image size 70x70, batch size 64, kernel size 5 and base filter size 64 with supercomputer

Figure 25 presents a bar chart of the data in Table 6, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 7. Image size 100x100 pixels, epochs 30, batch size 64, kernel size 5, base filter size 64

Pickle file count	**Image count**	**Images size (Gb)**	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**
100	20100	1.1	N/A	N/A	1050	1067
300	60300	3.3	N/A	N/A	3339	3392
500	100460	5.4	N/A	N/A	5774	5869

Table 7 presents the results of running the CNN training on a laptop and a supercomputer with varying Pickle file counts. The image size is given as 100x100 pixels. In the training, 30 epochs are used, the kernel size is set to 5, and the batch size and base filter are set to 64. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training. In the case of a pickle count of 100 and above, the times for the laptop are not applicable, as the laptop memory is incapable of loading the size of the images.

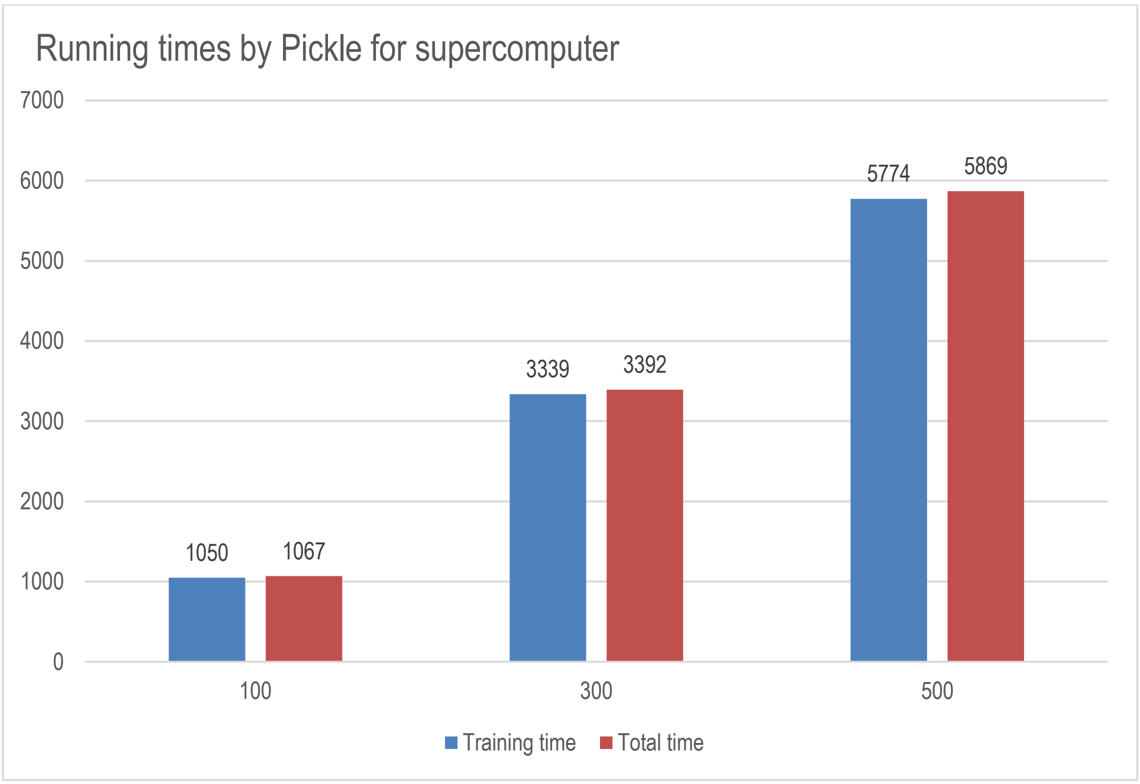


FIGURE 26. Running time in seconds by Pickle for image size 100x100, epochs 30, batch size 64, kernel size 5 and base filter size 64 with supercomputer

Figure 26 presents a bar chart of the data in Table 7, which compares only the total time spent on the supercomputer and the time spent on training. The figure clearly demonstrates that most of the execution time is spent on CNN training.

TABLE 8. Pickle count 50 epochs 10, batch size 32, kernel size 3, base filter size 32

Image size	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**

60x60	231	237	24	41
70x70	310	316	29	36
80x80	776	794	76	91
100x100	1107	1125	52	75

Table 8 presents the results of running the CNN training on a laptop and a supercomputer with varying image size. In the training, 50 Pickle files are used, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training.

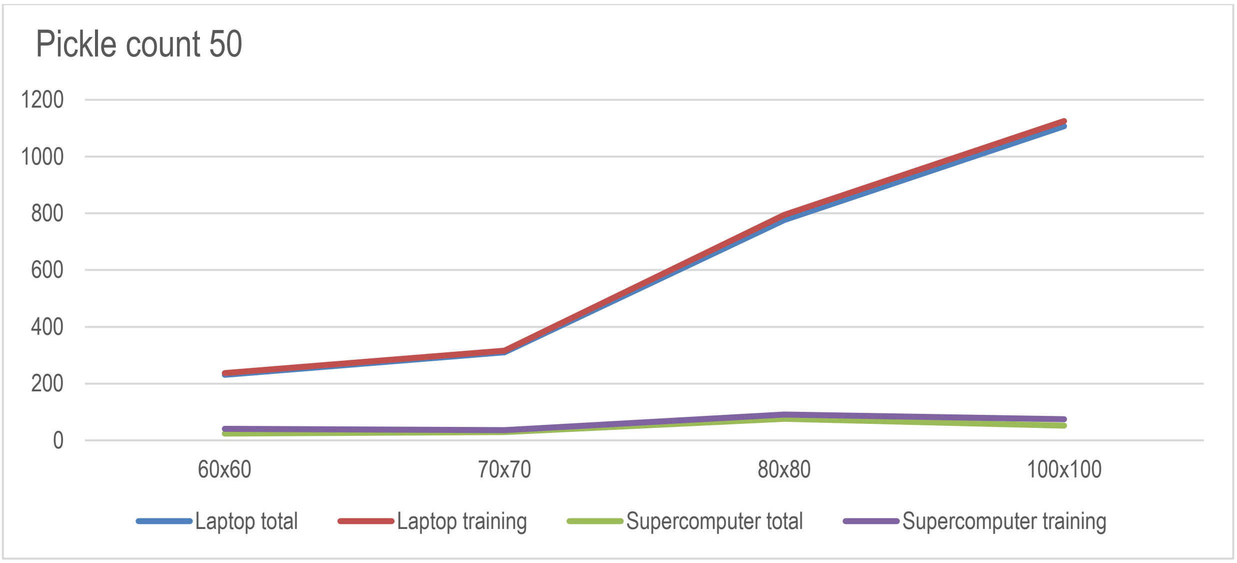


FIGURE 23. Running times in seconds by Pickle count 50 for image size 60x60, 70x70, 80x80 and 100x100

Figure 27 presents a line graph of the data in Table 8, which compares the total time spent on laptops and supercomputers with the time spent on training. The graph clearly demonstrates that the execution time begins to increase for laptops at the 70x70 image size. The supercomputer performs relatively consistently with all these image sizes when the training data size is the same.

TABLE 9. Pickle count 100, epochs 10, batch size 32, kernel size 3, base filter size 32

Image size	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**

60x60	552	566	43	67
70x70	563	574	57	69
80x80	2160	2221	76	91
100x100	2381	2478	105	137

Table 9 presents the results of running the CNN training on a laptop and a supercomputer with varying image size. In the training, 100 Pickle files are used, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training.

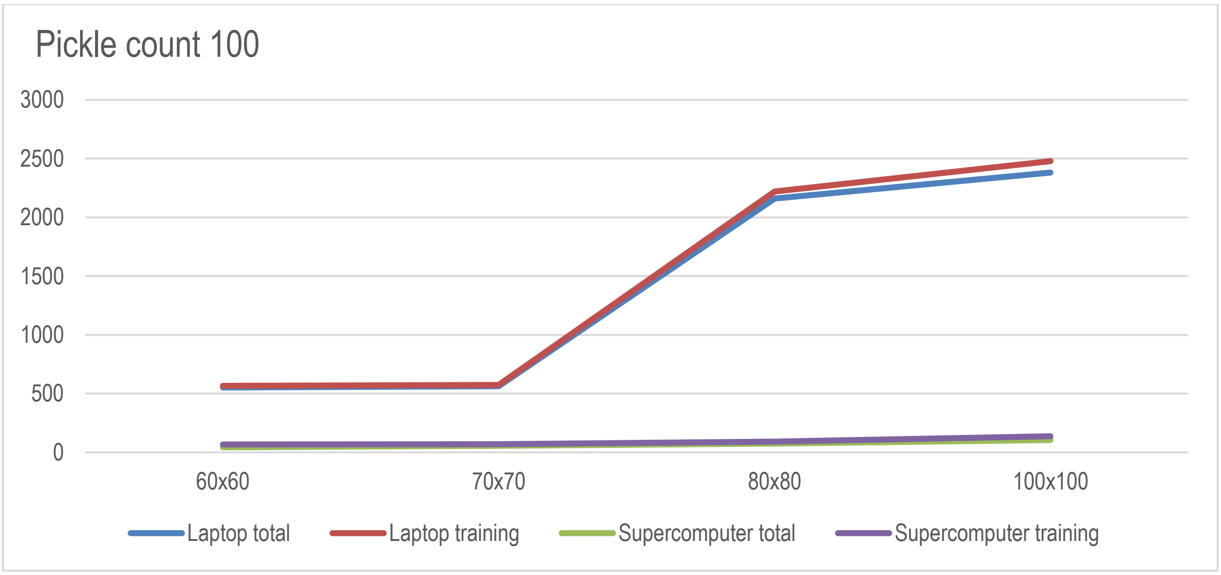


FIGURE 28. Running times in seconds by Pickle count 100 for image size 60x60, 70x70, 80x80 and 100x100

Figure 28 presents a line graph of the data in Table 9, which compares the total time spent on laptops and supercomputers with the time spent on training. The graph clearly demonstrates that the execution time begins to increase for laptops at the 70x70 image size but decrease after size 80x80. The supercomputer performs relatively consistently with all these image sizes when the training data size is the same.

TABLE 10. Pickle count 500, epochs 10, batch size 32, kernel size 3, base filter size 32

Image size	**Laptop training time [s]**	**Laptop total time [s]**	**Supercomputer training time [s]**	**Supercomputer total time [s]**
60x60	5514	5769	213	356
70x70	4666	5092	283	341
80x80	10612	11609	351	432
100x100	13905	14526	522	641

Table 10 presents the results of running the CNN training on a laptop and a supercomputer with varying image size. In the training, 500 Pickle files are used, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training.

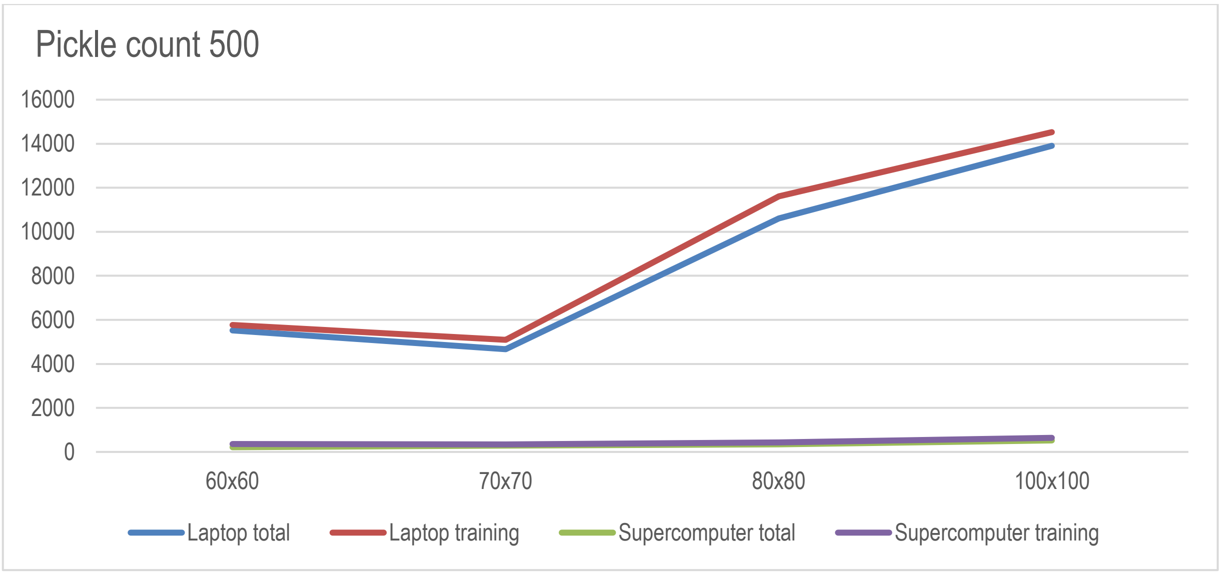


FIGURE 29. Running times in seconds by Pickle count 500 for image size 60x60, 70x70, 80x80 and 100x100

Figure 29 presents a line graph of the data in Table 10, which compares the total time spent on laptops and supercomputers with the time spent on training. The graph clearly demonstrates that the execution time begins to increase for laptops after the 70x70 image size. The supercomputer performs relatively consistently with all these image sizes when the training data size is the same.

TABLE 11. Pickle count 750, epochs 10, batch size 32, kernel size 3, base filter size 32

Image size	**Laptop training time [s]**	**Laptop total time [s]**	**Supercom-puter training time [s]**	**Supercom-puter total time [s]**
60x60	15444	17271	321	492
70x70	21179	23268	424	500
80x80	N/A	N/A	524	649

Table 11 presents the results of running the CNN training on a laptop and a supercomputer with varying image size. In the training, 750 Pickle files are used, 10 epochs are used, the kernel size is set to 3, and the batch size and base filter are set to 32. The table illustrates the total execution time of the program, which encompasses the uploading of images to the central memory and the elapsed time of CNN training. Furthermore, the board depicts the time exclusively dedicated to CNN training. In the case of an image size 80x80, the times for the laptop are not applicable, as the laptop memory is incapable of loading the size of the images.

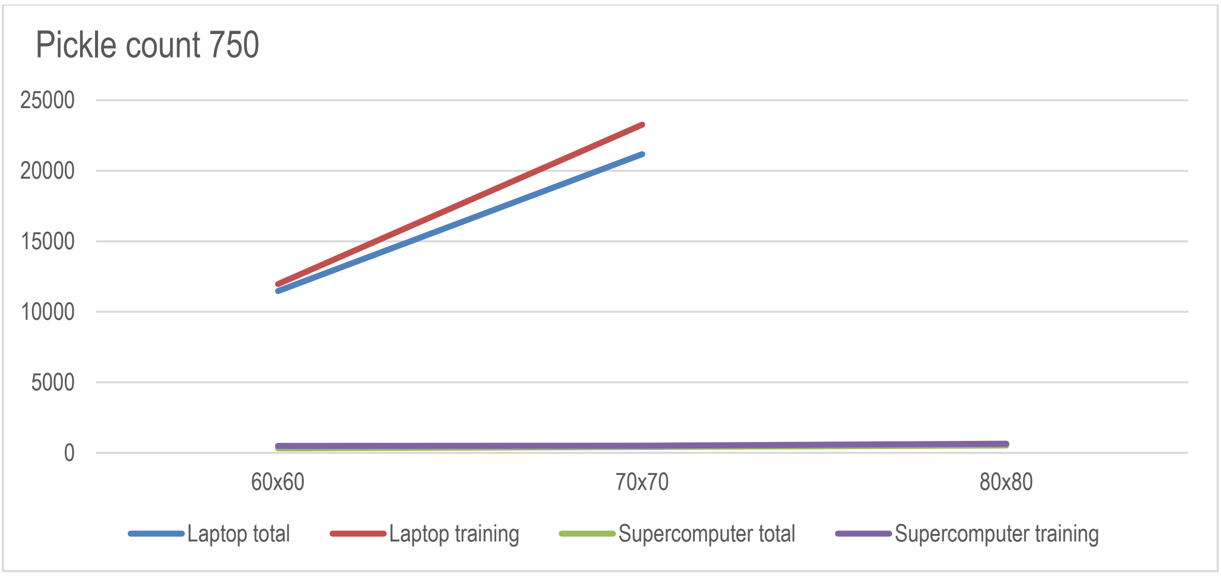


FIGURE 30. Running times in seconds by Pickle count 750 for image size 60x60, 70x70, 80x80

Figure 30 presents a line graph of the data in Table 11, which compares the total time spent on laptops and supercomputers with the time spent on training. The graph clearly demonstrates that the execution time begins to increase for laptops. After the 70x70 image size, the laptop is unable to produce a result due to its limited memory. The supercomputer performs relatively consistently with all these image sizes when the training data size is the same.

5 DISCUSSION AND CONCLUSIONS

In this study, the examination of the supercomputer settings revealed the existence of multiple optimization configurations, resulting in a considerable range of performance outcomes. It was determined that optimizing performance was not a viable approach in this study; instead, a setting was selected that would achieve performance results exceeding 600 GFLOPS with a minimal queue time. In contrast, on a laptop, the performance was significantly lower at 70 GFLOPS. The disparity in these figures is not as pronounced as it could be. According to theoretical calculations, the maximum processing capability of a supercomputer should be approximately 1.8 PFLOPS, while a typical laptop should be capable of approximately 108.8 GFLOPS. Consequently, a supercomputer is theoretically capable of reaching approximately 17000 times greater processing power, whereas in actuality, its performance has only reached approximately 9 times greater in this study. Reason so much lower result is that in this study we didn’t optimize code and running environment. Instead run as basic setting like normal non-professional person would do at the beginning. Better performance metrics for the supercomputer were obtained, but the queue time increased for them. Only 10000 x 10000 random matrix multiplication could be run reasonably on the laptop. Numbers higher than that loaded the machine so much and for so long that it didn't make sense to calculate them.

The algorithm utilized to calculate the flops may not be the optimal choice for that specific calculation. However, it is straightforward and readily executable across diverse computing platforms, requiring only a functional Python environment. There is no necessity for additional tools to be installed, and it can be executed within the terminal. The algorithm was subjected to a comparative analysis with a third-party tool, Linx, on a laptop computer, and the outcomes were found to be comparable. This tool can be employed to ascertain the relative efficiency of the supercomputer in comparison to the laptop. It can also be used to identify the optimal values for running the instructional material on the machine in order to achieve the desired level of efficiency and run time.

The collection of image data for CNN teaching was a time-consuming process that occupied the computer for several days. To ensure the validity of the recorded data, it was necessary to adjust the webcam to focus on the correct area of the monitor where the simulated signal was displayed and to study the recorded data. The utility program that was written for opening pickle files was of great assistance in this regard. In future studies, it would be beneficial to have an easier way to

separate the simulated signal from the simulation application UI, with the necessary metadata, for example, by displaying it in a separate window.
The primary objective of the study was to ascertain the circumstances under which it would be advantageous to employ a supercomputer for machine learning instead of a regular computer. That was examining how the impact of varying parameters on the teaching time when CNN instruction was executed, with the most crucial parameters being the size of the images utilized for teaching and the number of teaching images. Additionally, a few independent tests were conducted to assess the efficacy of different epochs, kernel sizes, and batch sizes when utilized with a regular computer and a supercomputer.

From the measurement results obtained, the following observations were made.

The greatest impact on performance times was observed when the pixel size of the teaching image was increased. On both machines, performance times were naturally much lower for small (60x60) pixel values than for large values (100x100). The supercomputer was found to take a clearly shorter time to execute the tutorial scripts than a normal computer. The larger the image size, the more efficient the teaching time on the supercomputer, with the supercomputer being found to be faster than the regular computer. At the smaller size, the times were almost identical. However, at the larger size, the time required for teaching was more than 20 times fast on the supercomputer. Laptop runtimes significantly increased when the image size was raised, and when it was 70 pixels and over. Supercomputer handles tested image size increasing pretty stability and we didn't observe as big impact with it that it was with laptop. When the image count was raised, we observed that both computer times increased nearly linearly. This is due to an increase in the memory allocation required for loading the images to memory.

It is evident that the number of images has a significant impact on the time required to complete the task. The supercomputer is capable of storing a considerably larger number of training images. This limitation of conventional computers should also be taken into account when contemplating a transition to supercomputers. It is evident that the supercomputer is capable of processing large amounts of data at a faster and more efficient rate. It is important to note that transferring learning data images, which are typically large amounts of data, to a supercomputer does require time. However, if there is no need to edit or change them, the time taken is not a significant concern, given that images are usually loaded in the background and data transfer speeds are now reasonably good. It is likely that a few hours to tens of gigabytes of data can be transferred.

In these studies, we utilize a limited number of parameters for image count and size due to the constraints of laptop memory capacities. For subsequent studies, it is necessary to modify both parameters with smaller increments and significantly larger values. This approach enables the identification of the optimal cap when transitioning from laptop to supercomputer calculations.

The primary issue with a supercomputer that results in unnecessary waiting is the queuing time of resources. On a supercomputer, a process may have to wait an indefinite amount of time to run. The longest time for in this work was 1 day and 15 hours and 44 minutes. Typical queuing time for more efficient computing power was up to 3 hours, while very basic computing powers may run immediately. In contrast, on a normal machine, the run commences immediately. This is one of the factors to be taken into account when considering transferring the work to a supercomputer.

The decision to transition to a supercomputer should be made when the need arises to utilize larger image sizes in educational materials, exceeding tens of gigabytes, or when the necessity arises to employ dozens of epochs. In general, if teaching on a conventional computer requires half of a typical workday and the resources of a supercomputer are available as an alternative, then a transition to a supercomputer should be strongly considered. The fundamental computing capacity of supercomputers remains considerably superior to that of conventional computers. Consequently, processing commences almost instantaneously, and the requisite calculations are completed in a few minutes, whereas with a conventional computer, the same calculations would take considerably longer, often spanning several tens of minutes. Another factor to be taken into account when utilizing a conventional computer is that the processing of calculations places a significant demand on the processor and, as a result, the computer may become inaccessible for other tasks during the calculations process.

The research thesis writer reached the conclusion that when the time required to run calculations exceeds half an hour, the process would be good migrated to a supercomputer. This necessitates the initial configuration of the supercomputer, including the transfer of data and scripts and the configuration of the operating environment. However, subsequent modifications to the script and the execution of tasks are relatively straightforward and rapid. The utilization of the supercomputer allows the working computer to be employed for other tasks.

REFERENCES

Aksela, M., Marchal, S., Patel, A., Rosenstedt, L. & WithSecure 2022. The security threat of AI-enabled cyberattacks. Accessed 19.4.2024. Traficom 2024. https://www.traficom.fi/sites/default/files/media/publication/TRAFICOM_The_security_threat_of_AI-enabled_cyberattacks%202022-12-12_en_web.pdf

Anant 2023. Understanding FLOPS: From Teraflops to Exaflops in High-Performance Computing. Medium 2024. Accessed 1.5.2024. https://medium.com/@anant3104/understanding-flops-from-teraflops-to-exaflops-in-high-performance-computing-7e9f2fd49b62

Crabtree, Matt 2023. Datacamp 2024. Accessed 23.4.2024. https://www.datacamp.com/blog/what-is-machine-learning

Cisco 2024. What is Artificial Intelligence in Networking? Accessed 19.4.2024. https://www.cisco.com/c/en/us/solutions/artificial-intelligence/artificial-intelligence-machine-learning-in-networking.html

CSC 2019. Supertietokone Puhti on avattu tutkijoiden käyttöön. Accessed 23.4.2024. https://www.csc.fi/-/supertietokone-puhti-on-avattu-tutkijoiden-kayttoon

European parliament 2023. What is artificial intelligence and how is it used? Accessed 13.7.2023. https://www.europarl.europa.eu/topics/en/article/20200827STO85804/what-is-artificial-intelligence-and-how-is-it-used

Gill, Jagreet Kaur 2023. Artificial Intelligence in Edge Computing | Benefits and Use-Cases. Xenonstack 2024. Accessed 19.4.2024. https://www.xenonstack.com/blog/ai-edge-computing

Hintsala, Juha 2023. 6G-signaali tietää kohta, miten voit kotona – tästä on kyse biosignaalien mittaamisessa. Yle 2023. Accessed 18.3.2023. https://yle.fi/a/74-20015687

IBM 2024. What is edge AI? Accessed 20.4.2024. https://www.ibm.com/topics/edge-ai

Insta 2023. Neuroverkot analytiikan edistäjinä. Accessed 18.3.2023. https://www.insta.fi/ajankohtaista/neuroverkot-analytiikan-edistäjinä

Intel 2024. App Metrics for Inter Microprocessors. Accessed 23.4.2024. https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf

Intel 2024. Intel Core i7-8550U Processor. Accessed 23.4.2024. https://ark.intel.com/content/www/us/en/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html

Juniper 2024. What is artificial intelligence for networking? Accessed 23.4.2024. https://www.juniper.net/us/en/research-topics/what-is-ai-for-networking.html

Lutkevich, Ben 2022. Supercomputer. TechTarget 2024. Accessed 1.5.2024. https://www.techtarget.com/whatis/definition/supercomputer

Matlis, Jan 2005. Sidebar: The Linpack Benchmark. Computer World 2024. Accessed 1.5.2024. https://www.computerworld.com/article/1715475/sidebar-the-linpack-benchmark.html

Nokia. 6G explained. Accessed 18.3.2023. https://www.nokia.com/about-us/newsroom/articles/6g-explained/

Räisänen, Antti & Lehto, Arto 2001. Radiotekniikan perusteet. 10. Uudistettu painos. Helsinki: Yliopistokustannus/Otatieto. Hakapaino Oy.

Seraydari, L., Mosinyan, A. & Kotolyan, A. 2023. PlatAi. Accessed 19.4.2024. https://plat.ai/blog/ai-powered-network-optimization-in-telecommunication/

Soft for Bro 2024. LINX (LinPack). Accessed 20.4.2024. https://soft4bro.com/soft/linx-linpack-download

Tampereen yliopisto 2023.Tekoälyn historia. Kaupunkiseudun ihmiskeskeiset tekoälyratkaisut (KITE). Accessed 13.7.2023. https://projects.tuni.fi/kite/tekoalysta-yleisesti/tekoalyn-historia/

Tensorflow 2024. Data augmentation. Accessed 23.4.2024. https://www.tensorflow.org/tutorials/images/data_augmentation

Vadapalli, Pavan 2024. Python Tutorial. Upgrad 2024. Accessed 1.5.2024. https://www.upgrad.com/tutorials/software-engineering/python-tutorial/why-python-is-interpreted-language/

Van Heerdan, Hannes 2023. The impact of ai in telecommunications. Telecoms 2023. Accessed 19.4.2024. https://telecoms.adaptit.tech/blog/the-impact-of-ai-in-telecommunications/#Innovation_in_the_Telecom_Industry

Zieniūtė, Ugnė 2022. Mitä on edge computing eli reunalaskenta. NordVPN 2022 Accessed 18.3.2023. https://nordvpn.com/fi/blog/reunalaskenta/

APPENDICES

```
import tkinter as tk
from tkinter import filedialog
import pickle
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

class PickleViewer:
    def __init__(self, root):
        self.root = root
        self.root.title("Pickle Data Viewer")

        # create widgets
        self.file_label = tk.Label(self.root, text="No file selected.")
        self.select_file_button = tk.Button(self.root, text="Select file",
command=self.select_file)
        self.index_entry = tk.Entry(self.root, width=10)
        self.go_to_button = tk.Button(self.root, text="Go to index", com-
mand=self.go_to_index)
            self.prev_button  =  tk.Button(self.root,  text="Prev",  com-
mand=self.prev_image)
            self.next_button  =  tk.Button(self.root,  text="Next",  com-
mand=self.next_image)
        self.canvas = FigureCanvasTkAgg(plt.figure(), self.root)

        # add new widgets for text fields
        self.info_label = tk.Label(self.root, text=f'Saato X: | Saato Y: |
Saato teho: | Laatu: | Aika:')

        # layout widgets
        self.file_label.pack()
        self.select_file_button.pack()
        self.index_entry.pack(side=tk.LEFT)
        self.go_to_button.pack(side=tk.LEFT)
        self.prev_button.pack(side=tk.LEFT)
        self.next_button.pack(side=tk.LEFT)
        self.canvas.get_tk_widget().pack(side=tk.BOTTOM)

        # add new widgets to the layout
        self.info_label.pack(side=tk.LEFT)

        # initialize variables
        self.file_path = None
        self.data = None
        self.current_index = 0
    def select_file(self):
        self.file_path = filedialog.askopenfilename(title="Select a pickle
file", filetypes=[("Pickle files", "*.p")])
        if self.file_path:
            self.file_label.config(text=self.file_path)
            with open(self.file_path, "rb") as f:
                self.data = pickle.load(f)
            self.show_image()

    def go_to_index(self):
        try:
            index = int(self.index_entry.get())
           if 0 <= index < len(self.data["opetus"]["vastaanotinmatriisi"]):
                self.current_index = index
                self.show_image()
        except ValueError:
            pass

    def prev_image(self):
        if self.current_index > 0:
            self.current_index -= 1
            self.show_image()

    def next_image(self):
         if self.current_index < len(self.data["opetus"]["vastaanotinmat-
riisi"]) - 1:
            self.current_index += 1
            self.show_image()

    def show_image(self):
        if self.data:
             image = self.data["opetus"]["vastaanotinmatriisi"][self.cur-
rent_index]
            label = f'{self.data["opetus"]["leima"][self.current_index]} |
index: {self.current_index}'
            plt.clf()
            plt.imshow(image)
            plt.title(label)
            self.canvas.draw()
            self.info_label.config(text=f'''
               Saato X: {self.data["opetus"]["saatoparametri_x"][self.cur-
rent_index]} | Saato Y: {self.data["opetus"]["saatoparametri_y"][self.cur-
rent_index]}     |     Saato     teho:     {self.data["opetus"]["saatopara-
metri_teho"][self.current_index]}       |       Laatu:       {self.data["ope-
tus"]["laatu"][self.current_index]}  |  Aika:  {self.data["aika"][self.cur-
rent_index]} ''')
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = PickleViewer(root)
    root.mainloop()
```

```
import numpy as np
import time
import psutil
import platform

'''
GFLOPS (GigaFLOPS): Measures computing power in terms of billions of float-
ing-point operations per second.
The GFLOPS metric is commonly used in computer graphics and scientific
computing.
'''

# read processor information
cpu = platform.processor()
print(f'Prosessor: {cpu}')
# memory
mem = psutil.virtual_memory()
print(f'Memory: {mem.total / 1e9}, GB')

# Generate random matrix
n = 10000

# repeats
repeats = 1

# Multiply matrix and show time
start_time = time.perf_counter()

for i in range(repeats):
    print(f'round {i}. ({(i/repeats)*100:.2f} %)', end="\r")
    A = np.random.rand(n, n)
    B = np.random.rand(n, n)
    C = np.dot(A, B)

print(f'round {i+1}. (100 %)', end="\n")
end_time = time.perf_counter()

# time per multiplication
total_time = end_time - start_time
time_per_dot = total_time / repeats

# calculate GFLOPS
gflops = (2 * n ** 3 - n ** 2) / (total_time * 1e9)
```

```
# Print result
print(f'Time per multiplication: {time_per_dot:.2f} second')
print(f'Permormance: {gflops:.2f} GFLOPS')
print(f'Total time: {total_time}')
```

```
from cnn_opetus_class import CnnLearning
from cnn_testi_class import CnnTesting
import time
from common import Common

class Compined:
    def __init__(self):
        print("run cnn learn and tests")

if __name__ == '__main__':
    start_time = time.perf_counter()

    img_size=100
    file_count=750
    epochs=25
    batch_size=32
    kernel_size=3
    base_filter_size=32

     print(f'image  size:  {img_size},  file  count:  {file_count}  epochs:
{epochs} batch size: {batch_size} kernel size: {kernel_size} base filter
size: {base_filter_size}')
     cl  = CnnLearning(files='../aineistokansio/opetus', img_size=img_size,
file_count=file_count, epochs=epochs)
    learn_data = cl.load_learning_data()
    cl.format_data(learn_data=learn_data)
    cl.create_model()
    cl.compile_model()
    model_path = cl.fit_model()
    cl.evaluate_model()

    end_time = time.perf_counter()
    time_total =  end_time - start_time
    print("Time Whole Trainging")
    Common.print_time(Common, time_total)

    start_time = time.perf_counter()
     ct  =  CnnTesting(files='../aineistokansio/testi',  img_size=img_size,
file_count=20)
    ct.set_model_path(model_path)
    ct.load_model()
    ct.run_test()

    end_time = time.perf_counter()
    time_total =  end_time - start_time
    Common.print_time(Common, time_total)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import time
import random
import numpy as np

from common import Common

class CnnLearning:
     def  __init__(self,  files='../aineistokansio/opetus',  img_size=100,
file_count=2, epochs=10, batch_size=32,
                 kernel_size=3, base_filter_size=32):
        print("CnnLearning Init")
        self.common = Common(img_size=img_size, file_count=file_count)
        self.files = files
        self.img_size = img_size
        self.file_count = file_count
        self.LEIMAT = self.common.get_labels()
        self.learn_imgs = []
        self.learn_labels = []
        self.learn_data = []
        self.X_train = []
        self.X_val = []
        self.y_train = []
        self.y_val = []
        self.history = None
        self.epochs = epochs
        self.batch_size = batch_size
        self.kernel_size = kernel_size
        self.base_filter_size = base_filter_size

    def set_epochs(self, epochs):
        self.epochs = epochs

    def set_batch_size(self, size):
        self.batch_size = size

    def set_kernel_size(self, size):
        self.kernel_size = size

    def set_base_filter_size(self, size):
        self.base_filter_size = size
    def load_learning_data(self):
                   self.learn_imgs,    self.learn_labels    =    self.com-
mon.get_pickle_data_from_files(
            self.files, self.file_count)
        self.learn_data = self.common.create_images_and_labels_data(
            self.learn_imgs, self.learn_labels)
        random.shuffle(self.learn_data)
        return self.learn_data


    def format_data(self, learn_data, test_size=0.3):
        X = []
        y = []

        for features, label in learn_data:
            X.append(features)
            y.append(label)

         X  =  np.array(X).reshape(-1,  self.img_size,  self.img_size,  3)  /
255.0
        y =  tf.keras.utils.to_categorical(y, num_classes=5)

              self.X_train,   self.X_val,   self.y_train,   self.y_val   =
train_test_split(X, y, test_size=test_size)
        return self.X_train, self.X_val, self.y_train, self.y_val

    def create_model(self):
        # model
        self.model = Sequential()
              self.model.add(Conv2D(filters=self.base_filter_size,   ker-
nel_size=self.kernel_size,
                           activation='relu', input_shape=[self.img_size,
self.img_size, 3]))
        self.model.add(MaxPooling2D(pool_size=2, strides=2))
             self.model.add(Conv2D(filters=self.base_filter_size*2,   ker-
nel_size=self.kernel_size, activation='relu'))
        self.model.add(MaxPooling2D(pool_size=2, strides=2))
        self.model.add(Flatten())
        self.model.add(Dense(units=128, activation='relu'))
        self.model.add(Dense(5, activation='softmax'))
        self.model.summary()

    def compile_model(self):
        self.model.compile(loss='categorical_crossentropy', optimizer='ad-
am', metrics=['accuracy'])
```

```
    def fit_model(self):
        start_time = time.perf_counter()

        self.history = self.model.fit(self.X_train, self.y_train,
                                      batch_size=self.batch_size,
                                      epochs=self.epochs,
                                     validation_data=(self.X_val, self.y_val))

        end_time = time.perf_counter()
        time_total =  end_time - start_time
        self.common.print_time(time_total, " opetus")

                                                          model_name          =
f'IS_{self.img_size}E_{self.epochs}BS_{self.batch_size}LFC_{self.file_cou
nt}_model.h5'
        self.model.save(f'{model_name}')
        print(f'Saved model: {model_name}')
        return model_name

    def evaluate_model(self):
        print("-"*50, " EVALUATION ", "-"*50)

        result = self.model.evaluate(self.X_val, self.y_val)
        print("Test result:")
        print("Loss: ", result[0])
        print("Accuracy: ", result[1])

        print("-"*100)

if __name__ == "__main__":

    cl = CnnLearning(epochs=2)

    learn_data = cl.load_learning_data()
    cl.format_data(learn_data=learn_data)
    cl.create_model()
    cl.compile_model()
    cl.fit_model()
    cl.evaluate_model()
```

```
import tensorflow as tf
import time
import random
import numpy as np
import cv2
from common import Common

class CnnTesting:
          def    __init__(self,    model_path='IS100E3BS32LFC5_model.h5',
files='../aineistokansio/testi',
                 img_size=100, file_count=2):
        print("CnnTesting init")
        self.common = Common(img_size=img_size, file_count=file_count)
        self.files = files
        self.img_size = img_size
        self.file_count = file_count
        self.LEIMAT = self.common.get_labels()
        self.test_imgs = []
        self.test_labels = []
        self.test_data = []
        self.model_path = model_path

    def load_testing_data(self, file_path):
                    self.test_imgs,    self.test_labels    =    self.com-
mon.get_pickle_data_from_file(file_path)
        return self.test_imgs, self.test_labels

    def load_model(self):
        self.model = tf.keras.models.load_model(self.model_path)

    def reshape_image(self, data):
        new_array = cv2.resize(data, (self.img_size, self.img_size))
        return new_array.reshape(-1, self.img_size, self.img_size, 3)

    def set_model_path(self, model_path):
        self.model_path = model_path

    def run_test(self):
        # predict test data
        test_counter = 0
        correct_counter = 0
        invalid_counter  = {}

        '''load list of teset files'''
        file_list = self.common.load_folder(self.files)
        start_time = time.perf_counter()
        '''select randomly files by given count'''
        for _ in range(self.file_count):
            x = random.choice(file_list)
            file_list.remove(x) # remove used one from list

                          test_imgs,    test_labels    =    self.load_test-
ing_data(f'{self.files}/{x}')

            for i in range(len(test_labels)):
                 # self.common.print_same_line(f'{i+1}/{len(test_labels)}
({(i+1)/len(test_labels)*100:.2f} %)')
                      prediction  =  self.model.predict([self.reshape_im-
age(test_imgs[i])], verbose=0)

                test_counter += 1
              if  self.LEIMAT[test_labels[i]] == np.argmax(prediction[0]):
                    correct_counter += 1
                else:
                        invalid_counter[i]  =  f'file:  {x}  tod:  {self.LEI-
MAT[test_labels[i]]} - enn: {np.argmax(prediction[0])}'

        end_time = time.perf_counter()
        time_total = end_time - start_time

        print("", end='\n')
        print("Time testing total: ", time_total)
        self.common.print_time(time_total, " (testing total)")
        print("="*50)
        print(f"Wrongly predict: ")
        for i in invalid_counter:
            print(invalid_counter[i])
        print("_"*50)
           print(f"How  many  got  right:  {correct_counter}/{test_counter}
({(correct_counter/test_counter)*100:.2f}%)")


if __name__ == "__main__":
    ct = CnnTesting()

    ct.load_model()
    ct.run_test()
```

```
import os
import sys
import random
import pickle
import cv2

class Common:
    def __init__(self, img_size=100, file_count=2):
        self.img_size = img_size
        self.file_count = file_count
        self.LEIMAT = {
            "lepo": 0,
            "ylos": 1,
            "alas": 2,
            "oikea": 3,
            "vasen": 4,
        }

    def print_same_line(self, line):
        sys.stdout.write('\r' + ' ' * len(str(line)) + '\r')
        print(line, end='\r')

    def print_time(self, seconds_total, args=''):
        # Calculate time in days, hours, minutes, and seconds
        days, remainder = divmod(seconds_total, 86400)
        hours, remainder = divmod(remainder, 3600)
        minutes, seconds = divmod(remainder, 60)

        # Print results
          print(f'Total time{args}: {seconds_total} seconds ({int(days)}
days, {int(hours)} hours, {int(minutes)} minutes, {int(seconds)} seconds)')

    def change_bytes(self, size):
        """change bytes KB, MB, GB or TB format"""
        suffixes = ["B", "KB", "MB", "GB", "TB"]
        idx = 0
        while size >= 1024 and idx < len(suffixes) - 1:
            size /= 1024
            idx += 1
        return f'{size:.1f} {suffixes[idx]}'
    def load_folder(self, dir):
        path_size = 0
        dir_list = os.listdir(dir)

        # count size of file insize folder
        for ele in os.scandir(dir):
            path_size += os.path.getsize(ele)

        print(self.change_bytes(path_size))
        return dir_list

    ''' get data from pickle'''
    def get_pickle_data_from_files(self, dir, count):
        imgs = []
        labels = []

        file_list = self.load_folder(dir)

         # fetch data
        files_size = 0
        for _ in range(count):
            x = random.choice(file_list)
            file_list.remove(x)

            file = os.path.join(dir, x)
            with open(file, "rb") as f:
                pickle_data = pickle.load(f)
            imgs += pickle_data['opetus']['vastaanotinmatriisi']
            labels += pickle_data['opetus']['leima']

            files_size += os.path.getsize(file)
        print(f'Picture size: {self.change_bytes(files_size)}')
        print(f'Picture count:', len(imgs))
        return imgs, labels

    '''get data from single file'''
    def get_pickle_data_from_file(self, path):
        imgs = []
        labels = []

         # fetch data
        files_size = 0
        with open(path, "rb") as f:
            pickle_data = pickle.load(f)
        imgs = pickle_data['opetus']['vastaanotinmatriisi']
        labels = pickle_data['opetus']['leima']

        files_size = os.path.getsize(path)
        return imgs, labels

    def create_images_and_labels_data(self, data_images, data_labels):
        data = []
        for i in range(len(data_images)):
            try:
                    new_pic  =  cv2.resize(data_images[i],  (self.img_size,
self.img_size))
                data.append([new_pic, self.LEIMAT[data_labels[i]]])
            except Exception as e:
                print(e)
                pass
        return data

    def get_labels(self):
        return self.LEIMAT
```