Technical University Of Crete

Electrical & Computer Engineering

Digital Image And Signal Processing Laboratory

Thesis

Automated Mosaic Tesserae Segmentation Using Artificial Intelligence Techniques

Author:
Charilaos Kapelonis

Thesis Committee:
Prof. Michalis Zervakis (Supervisor)
Prof. Aristomenis Antoniadis
Prof. Michail G. Lagoudakis

Chania, 2025

Πολυτεχνείο Κρήτης

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Εργαστήριο Ψηφιακής Επεξεργασίας Σήματος Και Εικόνας

Διπλωματική Εργασία

Αυτοματοποιημένη Κατάτμηση Ψηφίδων Μωσαϊκού με Χρήση Τεχνικών Τεχνητής Νοημοσύνης

Συγγραφέας:
Χαρίλαος Καπελώνης

Επιτροπή Διπλωματικής Εργασίας:
Καθ. Μιχάλης Ζερβάκης (Επιβλέπων)
Καθ. Αριστομένης Αντωνιάδης
Καθ. Μιχαήλ Γ. Λαγουδάκης

Χανιά, 2025

Abstract

Art is widely recognized as a reflection of civilization and mosaics represent an important part of cultural heritage. Mosaics are an old art form where a surface is decorated by attaching small pieces of material, called tesserae, to a base with adhesive. Due to their age and fragility, they are prone to damage, highlighting the need for digital preservation.

This thesis addresses the problem of digitalizing mosaics by segmenting the tesserae, producing a binary image in which they are clearly separated from the background. This task falls under the field of Image Segmentation, one of the primary tasks in Computer Vision. We aim to create a method that achieves it **automatically** and with **high accuracy**.

Although classical approaches on Image Segmentation yield decent results, we employ Deep Learning models and, more specifically, our work utilizes the latest Segment Anything Model 2 (SAM 2) by META AI, a foundation model that outperforms most conventional segmentation models. Our work involves manually annotating a set of mosaic images to create a dataset which we utilize both to fine-tune the model and to eventually evaluate its performance.

The quantitative results, including the evaluation metrics, show a significant improvement compared to the baseline SAM 2 model. The baseline SAM 2 model - evaluated on the *large* checkpoint - yields 89.00% IoU and 92.12% Recall, while our model achieves 91.02% IoU and 95.89% Recall on the same dataset. Furthermore, we show that our approach can yield even more impressive results if trained without computational and time limitations.

This task belongs to the field of Computer Vision problems, hence the qualitative results, such as visual comparative representations, are equally important. These visualizations further confirm that the fine-tuned model we employ brings us closer to an effective solution to this problem.

Keywords: Mosaics, Deep Learning, Computer Vision, Image Segmentation, SAM 2, Transfer Learning, Digital Preservation

Περίληψη

Η τέχνη αναγνωρίζεται ευρέως ως αντανάκλαση του πολιτισμού και τα ψηφιδωτά αντιπροσωπεύουν ένα σημαντικό κομμάτι της πολιτιστικής κληρονομιάς. Τα ψηφιδωτά είναι μια παλιά μορφή τέχνης όπου μια επιφάνεια διακοσμείται με την τοποθέτηση μικρών κομματιών υλικού, που ονομάζονται ψηφίδες, σε μια βάση με κόλλα. Λόγω της ηλικίας και της ευθραυστότητάς τους, είναι επιρρεπή σε φθορά, κάτι που αναδεικνύει την ανάγκη για την ψηφιακή τους διατήρηση.

Αυτή η εργασία αντιμετωπίζει το πρόβλημα της ψηφιοποίησης των ψηφιδωτών μέσω της κατάτμησης των ψηφίδων, παράγοντας μια δυαδική εικόνα στην οποία αυτές διαχωρίζονται καθαρά από το φόντο. Αυτή η εργασία εμπίπτει στον τομέα της Κατάτμησης Εικόνας, ένα από τα κυριότερα προβλήματα της Μηχανικής Όρασης. Στοχεύουμε στη δημιουργία μιας μεθόδου που το επιτυγχάνει **αυτόματα** και με **υψηλή ακρίβεια**.

Παρόλο που οι κλασσικές προσεγγίσεις στην Κατάτμηση Εικόνας παρουσιάζουν ικανοποιητικά αποτελέσματα, εμείς χρησιμοποιούμε μοντέλα Βαθιάς Μάθησης και, πιο συγκεκριμένα, η εργασία μας αξιοποιεί το πιο πρόσφατο μοντέλο Segment Anything Model 2 (SAM 2) της META AI, ένα θεμελιώδες μοντέλο που ξεπερνά τα περισσότερα κλασσικά μοντέλα κατάτμησης εικόνας. Η εργασία μας περιλαμβάνει χειροκίνητο annotation ενός συνόλου εικόνων ψηφιδωτών για τη δημιουργία ενός σετ δεδομένων, το οποίο χρησιμοποιούμε για τη βελτιστοποίηση του μοντέλου και την αξιολόγηση της απόδοσής του.

Τα ποσοτικά αποτελέσματα, συμπεριλαμβανομένων των μετρικών αξιολόγησης, δείχνουν σημαντική βελτίωση σε σύγκριση με το αρχικό μοντέλο SAM 2. Το αρχικό μοντέλο SAM 2 - αξιολογημένο στο *large* checkpoint - αποδίδει 89.00% IoU και 92.12% Recall, ενώ το δικό μας μοντέλο πετυχαίνει 91.02% IoU και 95.89% Recall στο ίδιο σετ δεδομένων. Επιπλέον, δείχνουμε ότι η προσέγγισή μας μπορεί να αποδώσει ακόμη πιο αξιοσημείωτα αποτελέσματα εάν εκπαιδευτεί χωρίς περιορισμούς υπολογιστικής ισχύος και χρόνου.

Το παρόν πρόβλημα ανήκει στον τομέα των προβλημάτων Μηχανικής Όρασης, επομένως τα ποιοτικά αποτελέσματα, όπως οπτικές συγκριτικές απεικονίσεις, είναι εξίσου σημαντικά. Αυτές οι απεικονίσεις επιβεβαιώνουν ότι το μοντέλο που χρησιμοποιούμε μετά τη βελτιστοποίηση μας φέρνει πιο κοντά σε μια αποτελεσματική λύση για αυτό το πρόβλημα.

Λέξεις-Κλειδιά: Ψηφιδωτά, Βαθιά Μάθηση, Μηχανική Όραση, Κατάτμηση Εικόνας, SAM 2, Μεταφορά Μάθησης, Ψηφιακή Διατήρηση

to my family, my girlfriend,
and my grandma, who left too soon...

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Michalis Zervakis, for accepting to supervise my thesis and for his continuous support and guidance.

I would also like to thank Dr. Marios Antonakakis for his valuable advice and insightful feedback throughout the development of this work.

Furthermore, I am grateful to Prof. Aristomenis Antoniadis for contributing the initial idea and for his warm encouragement, as well as to Prof. Michail G. Lagoudakis for kindly agreeing to serve as a member of the thesis committee.

I would like to thank my family - especially my parents - for their unwavering support, and my siblings, whom I deeply admire and love.

Finally, I want to thank my girlfriend, whose love and encouragement mean the world to me, and my close friends for always being there for me.

Contents

List of Figures

List of Tables

Introduction

1.1 Motivation

Cultural heritage is a reflection of society and preserving it is important for cultural diversity, growing a sense of belonging and imparting knowledge to the future generations. [1] Artworks are tangible expressions of cultural heritage and mosaics are a specific form of this art. **Mosaics** consist of small tiles (called tesserae) which are glued together to form 2D or even 3D patterns. This art form has existed for thousands of years and many modern artists continue to adapt this style.

The long history of mosaics highlights the importance of documenting and preserving these artworks. Due to their particular structure and materials, mosaics are highly vulnerable to deterioration over time, often suffering catastrophic damage. For these reasons, the **digitization of mosaics** is a crucial task. The classical method of digitizing mosaics involved manually outlining the individual tesserae, which, although accurate, is extremely time-consuming and requires expert knowledge, making it impractical for large-scale projects. To address this, recent studies, such as the documentation of the Byzantine mosaics of the Nativity Church, have demonstrated how photogrammetry and non-invasive diagnostic analyses can produce reliable, high-resolution digital representations at both micro- and macro-scales, thereby supporting conservation and long-term management [2]. Nevertheless, such approaches remain labor-intensive and often lack the automation needed for scalability.

Algorithmic approaches based on classical image processing have also been widely explored. While such methods can achieve satisfactory results, they typically require long and complex pipelines, parameter tuning, and significant computational resources. This limits their applicability to large-scale or real-time scenarios, where a faster and more cost-effective solution is required. At the same time, accurate mosaic segmentation is fundamental for comparing stylistic techniques across regions and time periods, assisting in the dating of artworks, and identifying the cultural influences embedded in their design. Finally, digitized mosaics can be shared effortlessly with people around the world, allowing potential further study and appreciation of these detailed artworks.

To overcome these limitations, this thesis proposes the use of deep learning and image processing techniques for the automatic segmentation of mosaic tesserae. Unlike manual or classical algorithmic approaches, this methodology aims to pro-

vide an accurate and efficient solution that can support large-scale documentation efforts. The work is primarily motivated by a collaborative project with the Micromachining Manufacturing Modeling Lab from the Department of Production Engineering Management, where the ultimate goal is to segment tesserae in order to enable their 3D reconstruction. This process not only facilitates the physical replication of mosaics but also contributes to the preservation of these artworks.

1.2 Related Work

Despite the importance of cultural preservation, the task of segmenting mosaic images has been the case of study by only few researchers. Initial works of this field focused on recognizing and interpreting shapes and patterns in mosaics with image processing techniques. In 2004, the authors [3] presented an application of digital image processing in the analysis of medieval mosaic conservation to reveal original patterns, while the works [4], [5] proposed a retrieval system for Arabo-Moresque decor images based on mosaic representations.

The aforementioned studies paved the way to understanding the patterns of mosaics and several research efforts emerged aiming to reduce manual intervention while improving accuracy and scalability in the process of automatically detecting the tesserae of the mosaics. There are two basic approaches for the task of extracting the individual tesserae: Edge Detection and Image Segmentation. Edge-based methods rely on detecting gray-level discontinuities, in which the positions where the gray level changes sharply in an image are generally the boundaries of different regions. Edge Detection models, such as the *"Holistically-Nested Edge Detection"* [6], can highlight the outlines of tesserae, but they often have difficulty separating tiles when the gaps between them are similar in color to the tiles. Segmentation-based methods rely on classifying each pixel in the image as belonging to a specific tessera or to the background. Unlike Edge Detection, Segmentation can provide complete regions for each tessera, making it easier to analyze their shape, size, and arrangement.

This approach is particularly effective for mosaics with closely packed tiles, as it captures the full extent of each tessera, even when the gaps between them are narrow or the colors are similar. Overall, Image Segmentation is generally more effective than Edge Detection for mosaic analysis, as it produces complete tessera regions rather than just outlines, enabling more accurate measurements and automated processing.

Color plays a crucial role in analyzing mosaics. By accurately capturing the color of each tessera, we can gain insights into the artist's choices and the materials used, while also making it easier to define the boundaries of individual tiles. Using color information helps distinguish tesserae even when their shapes are irregular or the gaps between them are small [7]. Segmentation can also be approached in a layered way. First, we can identify large segments that represent major objects or regions in the mosaic (concept-level segmentation) [8]. Then, within these larger areas, we can focus on segmenting the individual tesserae. This two-step approach combines a broad understanding of the mosaic's overall composition with a detailed look at each tile, improving both accuracy and efficiency.

One of the initial approaches, proposed in 2008, was based on the Watershed Algorithm, which was aided by a tesserae-based k-means classification (yielding higher accuracy than a pixel-based strategy). The authors also proposed a method to estimate the main directional guidelines of tesserae in mosaics [9]. In 2016, a group of researchers proposed Deformable Models using Genetic Algorithms, in which each possible segmentation is represented as a set of quadrangles whose shapes and positions change, aiming to capture variations in mosaics' age and style [10].

In 2020, a Deep Learning (DL) technique based on a Convolutional Neural Network, U-Net, was proposed. The researchers trained the U-Net on a set of manually annotated mosaics and performed segmentation at pixel level, which yielded impressive results [11]. Building on this approach, in 2021, the Mo.Se. (Mosaic Segmentation) Algorithm was proposed. This algorithm combines the U-Net3 architecture with the Watershed Algorithm. According to the authors, the combination of learning-based methods with procedural ones enhances segmentation performance in terms of overall accuracy [12].

However, these methods still struggle with varied mosaic styles and often fail to handle damaged or incomplete tesserae, while relying on heavy computational resources.

1.3 Contribution

This work focuses on building a tool capable of automatically segmenting mosaic images with high accuracy. The foundation of this project lies upon utilizing the advancements in Machine Learning and - more specifically - the field of Image Segmentation.

We propose leveraging the latest foundation model of META AI, referred to as Segment Anything Model 2 (SAM 2), which showcases impressive results in segmentation tasks. To better adapt this model to mosaic images, we apply transfer learning and introduce modifications tailored to handle the specific challenges of mosaics, such as small tesserae. We modify the code to enable automatic point generation, removing the need for user input, and extend the META researchers' work by integrating this automatic generation with model inference, a capability not previously supported.

In order to support this training, we create and propose a state-of-the-art dataset consisting of mosaic images retrieved from public sources, which we manually annotated. The dataset is augmented to increase diversity, including variations in tessera count, style, resolution, and image transformations, and combines multiple sources to ensure robust learning.

Additionally, we analyze specific configurations for training and evaluating the models, providing the results of our experiments. The parameters are optimized using Bayesian Optimization on a subset of the data to identify the optimal configurations.

Finally, we make the code, dataset, and fine-tuned models publicly available to facilitate future research. We evaluate the performance of our model and compare it with previous works, demonstrating significant improvements and enhanced generalization capabilities.

1.4 Thesis Outline

Chapter 2 presents the fundamental theoretical background of Machine Learning, with a particular focus on the Image Segmentation task. This chapter covers classical techniques as well as modern deep learning approaches. Additionally, it discusses the state-of-the-art *Segment Anything* models developed by META AI.

In Chapter 3, we propose a novel method to automate the tesserae segmentation process in mosaics by fine-tuning the latest SAM 2 model. We evaluate the fine-tuned models under various configurations and compare their performance with two baseline checkpoints from the pre-trained model.

In Chapter 4 we highlights the advantages of our approach through extensive experiments and visualizations, while also diving into the details of the training and

testing process.

Finally, Chapter 5 summarizes the main contributions of this thesis and compares it to previous works. Furthermore, in this chapter we present the limitations while reviewing the proposed method and, eventually, outline directions for future work.

Theoretical Background

2.1 Artificial Intelligence

Artificial Intelligence (AI) is the study and application of computer systems or machines that exhibit qualities similar to the human brain, such as the ability to interpret and produce language in a human-like manner, recognize or create images, solve problems, and learn from data provided to them. [13]

The field of AI not only focuses on understanding but also on building such intelligent entities that specialize in a wide variety of problems and can compute how to act effectively and safely in these novel situations.

Currently, AI is used in a wide range of subfields, from general tasks like learning and reasoning to more niche problems, such as playing chess, proving mathematical theorems, self-driving cars, or medical diagnoses. AI is relevant to any task that requires intellectual capabilities, thus making it a universal solution in modern problems.

The definition of intelligence specifically varies, ranging from ”fidelity to human performance” to the more abstract concept of ”doing the right thing”. Intelligence is considered by some as the property of actually having internal thought processes and reasoning, while others focus on the results of intelligent behavior, viewing it as an external characterization. [14]

2.2 Machine Learning

Machine Learning (ML) is a field of study in AI concerned with the construction of computer programs that automatically improve with experience.

It grew out of AI but utilizes concepts and results from many fields, including statistics, philosophy, information theory, biology, cognitive science, computational complexity, control theory and, mainly, AI itself.

In recent years, many successful ML applications have emerged. Examples include fraud detection systems, such as those used by financial institutions to flag suspicious credit card transactions, recommendation algorithms that personalize

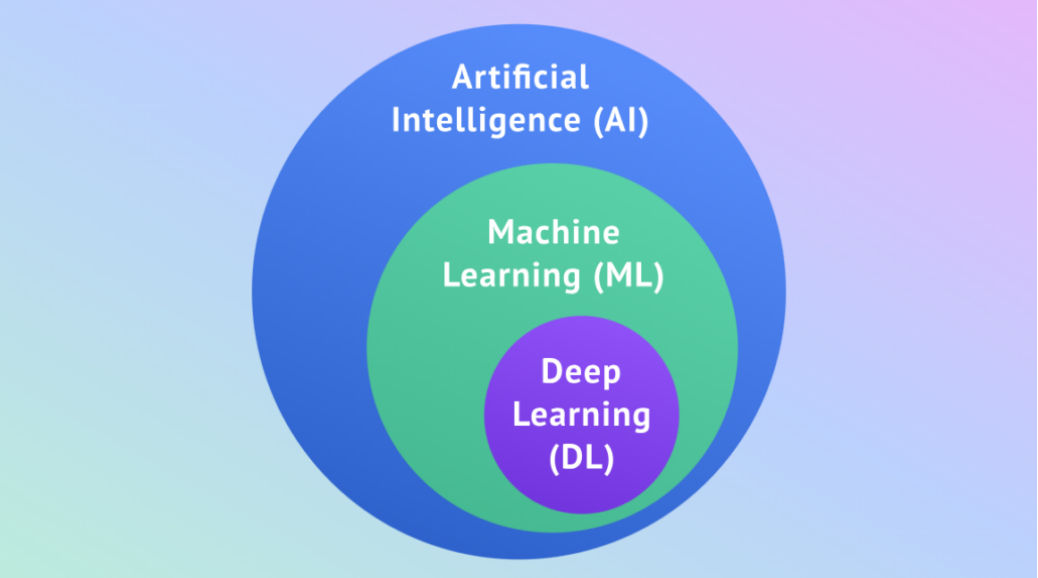


Figure 2.1: AI, ML and DL relation. Source: Datasaur.

users' reading preferences on platforms like Amazon or Netflix, and autonomous vehicles developed by companies like Tesla, which learn to navigate public highways. [15]

Another powerful application that has gained significant attention is the development of Large Language Models (LLMs), such as GPT-3 and BERT, which have demonstrated remarkable performance in natural language understanding and generation tasks.

At the same time, there have been radical advances in the theory and algorithms, with a team of researchers predicting that the impact of superhuman AI over the next years will be enormous, even bigger than that of the Industrial Revolution. [16]

In ML, the core concept revolves around solving a practical problem by algorithmically building a statistical model; this model is an important aspect of ML and has a deep mathematical background.

The process of building the model is called training and the ultimate goal for the model is to learn how to solve a general problem. [17]

2.3 Artificial Neural Networks

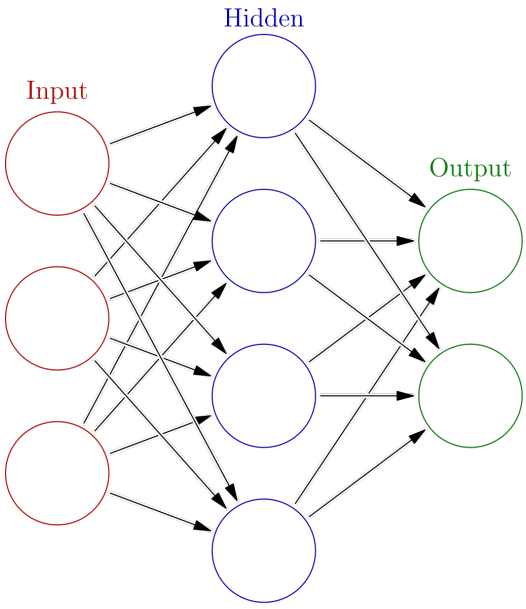


Figure 2.2: An example ANN with a hidden layer. Source: Wikimedia Commons.

Among the various approaches in ML, Artificial Neural Networks (ANNs) have gained significant attention due to their ability to model complex problems and produce impressive results, which makes them particularly effective for tasks like image recognition, natural language processing, computer vision, and more.

An ANN is a model of computation inspired by the structure of neural networks in the brain. Learning with neural networks was proposed in the mid-20th century. It yields an effective learning paradigm with impressive performances.

In simplified models of the brain, it consists of a large number of basic computing devices (neurons) that are connected to each other in a complex communication network, through which the brain is able to carry out highly complex computations.

Similarly, ANNs are computation constructs that are modeled after this computation paradigm. They are based on units called artificial neurons, which are aggregated into layers. [18]

2.3.1 Artificial Neuron

In biology, a neuron is a nerve cell that carries electrical impulses and serves as basic unit of our nervous system. It receives signals from other neurons through structures called dendrites, processes them in the cell body, and if the combined signal exceeds a certain threshold, it sends an output signal through the axon.

Similarly, in an artificial neuron:

- It receives inputs:
$$x_1, x_2, x_3, ..., x_n$$
- Inputs are associated with weights (reflecting the importance of each input):
$$w_1, w_2, w_3, ..., w_n$$
- A bias term b is added, making it possible to shift the final summation.
- The neuron computes a weighted sum of its inputs:
$$z = \sum_{i=1}^{n} w_i x_i + b$$
- Then, it applies an activation function $\phi(z)$ to the weighted sum to produce the output:
$$y = \phi(z)$$
This output can either serve as input to other neurons in subsequent layers or, if the neuron belongs to the output layer, it can represent the final output of the network.

The activation function introduces non-linearity into the model, allowing the network to learn and represent more complex relationships in data.

2.3.2 Network Architecture

A neural network consists of:

- An input layer that receives inputs, which can be the feature values of a sample of external data, such as images or documents.

- One or more hidden layers that process the inputs. These layers enable the network to learn complex patterns and abstract representations from the data.
- An output layer that produces the final prediction, such as recognizing an object in an image.

Although a single hidden layer with enough neurons can theoretically handle complex tasks, in practice, networks with multiple hidden layers - typically three or more - are more effective at capturing intricate relationships within data. These are known as Deep Neural Networks.

2.3.3 Learning Process

Learning in ANNs refers to the process by which the network adjusts its internal parameters - primarily weights and biases - to better perform a given task, based on metrics we have defined.

This process involves discovering patterns, representations, or strategies from data, and it can take various forms depending on the learning paradigm.

At the core of most learning methods lies the idea of optimizing a loss function, which quantifies how well the network performs.

Training typically involves three important concepts:

1. **Forward computation**: the network processes inputs through its layers.
2. **Performance evaluation**: based on the loss function we are evaluating how well the model performs.
3. **Backpropagation**: the network modifies its weights to reduce the loss.

Forward Computation

Forward computation refers to the process in which an input is propagated through the layers of a neural network to produce an output. During this phase, each neuron computes a weighted sum of its inputs, applies a bias, and passes the result through

an activation function. This transformation allows the network to model complex, non-linear relationships in the data.

The forward pass proceeds layer by layer, where the output of one layer serves as the input to the next, until the final output layer produces the network's prediction. This process is crucial for evaluating the network's performance and is the basis for subsequent optimization during training.

Activation Function

The activation function is a fixed, usually nonlinear function chosen before the learning is started. The fact that the activation function is nonlinear is important because, if it were not, any composition of units would still represent a linear function. The nonlinearity is what allows sufficiently large networks of units to represent arbitrary functions.

These are some of the most important activation functions used in neural networks and machine learning:

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic Tangent (Tanh):**

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Rectified Linear Unit (ReLU):**

$$\mathrm{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

Backpropagation

Backpropagation is the mechanism by which an ANN learns from its errors. After the forward pass produces an output, the network evaluates how far this output is from the expected result using a loss function.

For this process, usually the gradient of this loss with respect to each parameter in the network by applying the chain rule of calculus in reverse through the layers, hence the name gradient descent.

This process enables the network to understand which weights contributed most to the error and how they should be adjusted. By systematically propagating this information backward, the network updates its parameters to minimize future errors, effectively learning from the data.

This learning process relies on the availability of data with known outcomes, making it characteristic of supervised learning. Other learning paradigms adopt different strategies depending on the nature of the data and feedback available. [14]

2.3.4 Learning Paradigms

There are three main learning paradigms, which differ in the tasks they can solve and in how the data is presented to the model. Usually, the task and the data directly determine which paradigm should be used. Often, these paradigms can be used together in order to obtain better results.

Supervised Learning

Supervised Learning involves feeding pairs of inputs and their corresponding target output values into a learning algorithm, which adjusts the model's parameters so as to maximize the agreement between the model's predictions and the target outputs.

The outputs can either be discrete labels that come from a set of classes, or they can be a set of continuous, potentially vector-valued values. The first task is called classification, since we are trying to predict class membership, while the second is called regression, since historically, fitting a trend to data was called by that name.

Naturally, this type of learning requires an adequate amount of input data, highlighting the importance of data in the modern world.

Unsupervised Learning

In Unsupervised Learning, input data comes without labels, so the model learns patterns without any explicit feedback.

The most common Unsupervised Learning task is clustering: detecting potentially useful clusters of input examples, by grouping them based on their similarity to each other.

Another classic unsupervised task is called dimensionality reduction, which is used to reduce the number of variables in a dataset while trying to preserve some properties of the data, such as distances between examples.

Reinforcement Learning

Reinforcement Learning concerns learning how to act in an environment in order to maximize a cumulative reward signal. Unlike Supervised Learning, the agent is not provided with explicit instructions or correct actions; instead, it must discover effective behaviors through trial and error.

This interaction creates a closed-loop system in which the agent's actions influence future inputs and rewards. A big challenge is the requirement to balance exploitation (actions that are sure to bring reward) and exploration (trying new things with the risk of punishment) while pursuing long-term goals. [19] [20]

2.3.5 Overfitting and Underfitting

Overfitting means that the model predicts *too* well labels of the examples used during training, but fails to generalize making errors when presented with unseen data. It essentially fits too closely to the particularities of the training set.

Underfitting means that the model is too simplistic to capture the underlying structure of the data, performing poorly even on the training data. [17]

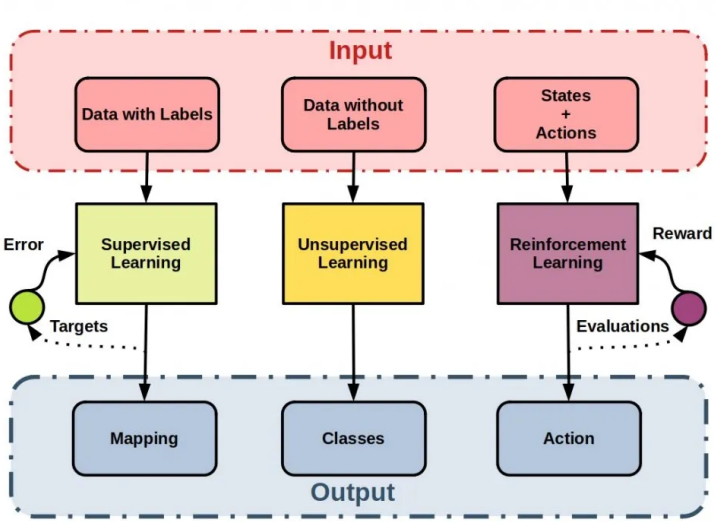


Figure 2.3: Learning paradigms visually explained. Source: Medium (@nimk).

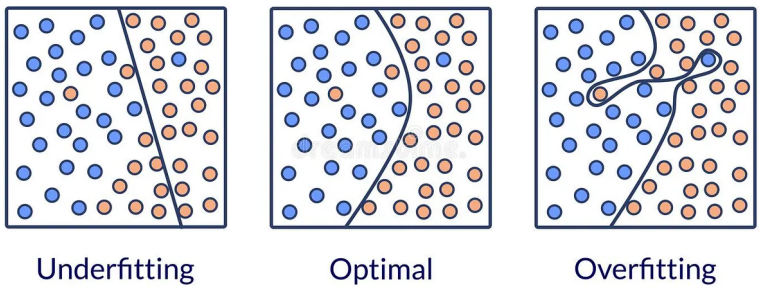


Figure 2.4: Illustration of overfitting and underfitting in ML. Source: AlliedOffsets Blog.

2.3.6 Data Augmentation

Insufficient labeled data is a common problem in ML, especially when dealing with more niche problems, where manual labeling is often required.

This problem is partially tackled through **data augmentation**.

The training set is expanded using transformations of the original data. For example, for problems that involve images as the training data, one can perform image augmentation to generate random images based on existing training data to improve the generalization ability of models. For images, these transformations typically refer to crops, rotations, additive noise, and more.

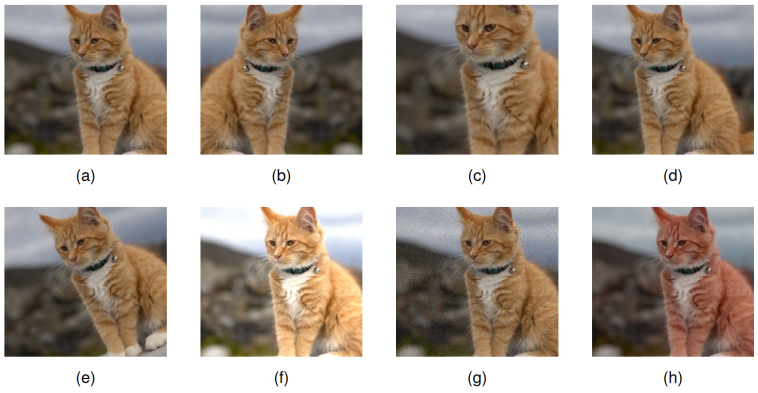


Figure 2.5: Illustration of data set augmentation, showing (a) the original image, (b) horizontal inversion, (c) scaling, (d) translation, (e) rotation, (f) brightness and contrast change, (g) additive noise, and (h) color shift. Source: [21].

In our case, the need for data augmentation is especially important due to the scarcity of annotated mosaic images and the high cost of manual labeling. Since the structure and appearance of mosaics can vary significantly in terms of style, color, and tesserae arrangement, a model trained on limited examples risks overfitting to specific patterns. By augmenting the dataset through controlled transformations, we introduce variability that helps the model learn more robust and generalizable features, ultimately improving its performance on unseen mosaics.

2.3.7 Optimization

In previous sections, the gradient computation of the loss function has been briefly mentioned. The core challenge following this step lies in applying an effective **optimization algorithm** to update the network's weights in a manner that ensures not only convergence but also strong generalization to unseen data.

One classical approach is gradient descent, which iteratively minimizes a differentiable multivariate function based on its gradient. However, such methods can easily get stuck in local minima or saddle points in high-dimensional spaces.

A widely adopted alternative is to introduce randomness, leading to the family of methods known as Stochastic Gradient Descent (SGD). Most modern implementations use variants of SGD that operate on subsets of data, known as minibatches, to estimate the loss and gradient. While basic SGD can suffer from noisy and unstable updates, enhancements like momentum address this by maintaining an exponentially decaying average of past gradients, resulting in more stable convergence. [20]

Over time, more advanced optimizers have emerged, including AdaGrad, RMSProp, and Adam, each improving aspects like learning rate adaptation and bias correction.

Today, Adam and its decoupled-weight-decay variant AdamW are among the most widely used optimizers due to their empirical effectiveness, although they still require careful hyperparameter tuning to perform optimally. [22]

2.3.8 Evaluation metrics

The performance of the network can be assessed using task-specific evaluation metrics, but the core concept of evaluation lies in True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN):

- TP: the instance is positive and it is classified (correctly) as positive
- TN: the instance is negative and it is classified (correctly) as negative
- FP: the instance is negative and it is classified (incorrectly) as positive
- FN: the instance is positive and it is classified (incorrectly) as negative

		Actual		
		Positive	Negative	Total
Prediction	Positive	TP	FP	$TP + FP$
	Negative	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	

Based on the confusion matrix, the following evaluation metrics are defined:

- **Accuracy**: the proportion of correctly classified instances among all instances:
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision**: the proportion of predicted positive instances that are actually positive:
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall (Sensitivity)**: the proportion of actual positive instances that are correctly predicted:
$$\text{Recall} = \frac{TP}{TP + FN}$$

Although the above metrics can provide insight on the performance of a network, one can use specific evaluation metrics, suitable for their task. For example, in image segmentation Intersection over Union, also called Jaccard Index, or Dice Coefficient can be used. [23]

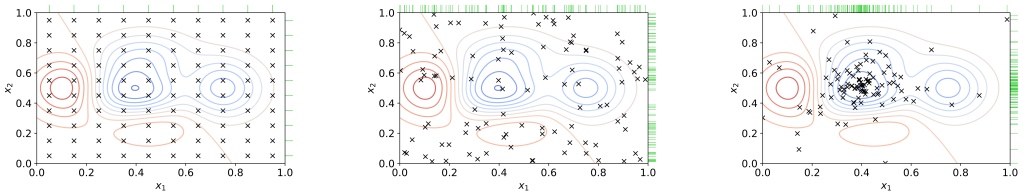
2.3.9 Hyperparameters

A very important aspect of building an effective model is hyperparameter tuning. The hyperparameters are the parameters that are used to either configure a model (e.g., learning rate to train a network) or to specify the algorithm used to minimize the loss function (e.g., the activation function and optimization techniques).

In order to fit a model into different problems, its hyperparameters must be tuned. Finding the right configuration has a direct impact on the performance of the model and often requires deep knowledge on the nature of the task. [24]

Some of the most common hyperparameter tuning techniques include:

- Grid Search
- Random Search
- Bayesian Optimization



(a) Grid Search (b) Random Search (c) Bayesian Optimizer

Figure 2.6: Common optimization techniques. Source: Wikimedia Commons.

2.3.10 Transfer Learning

Transfer Learning (TL) can be used to improve performance on some task A, for which training data is in short supply, leveraging data from a related task B, for which data is more plentiful. Tasks A and B should share the same type of inputs and exhibit some common underlying structure, exploiting the internal representation learned for one particular task (B) and, thus, enhancing the performance of task A.

In practice, this can be achieved by freezing the early layers of the network and fine-tuning the rest, meaning that the frozen layers keep the same parameters while the remaining parameters are readjusted on the dataset of task A. This is generally done with a very small learning rate for a limited number of iterations to ensure that the network does not overfit to the relatively small data set available for the new task. [21]

A common application of TL is found in ML-assisted medical diagnosis. For instance, a network pre-trained on a large, labeled dataset of everyday objects for an image segmentation task can be repurposed for segmenting medical images, such as skin lesions or radiographic scans.

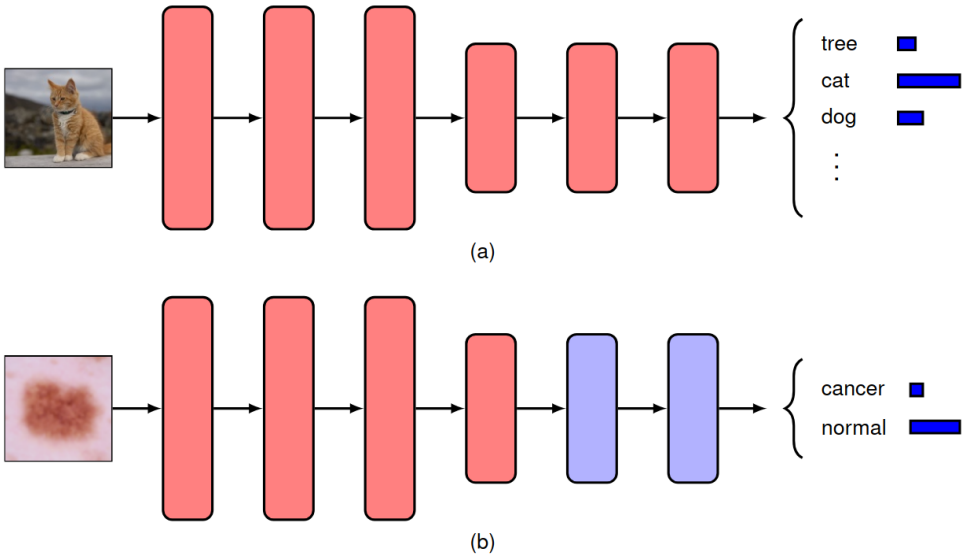


Figure 2.7: Schematic illustration of transfer learning. (a) A network is first trained on a task with abundant data, such as object classification of natural images. (b) The early layers of the network (shown in red) are copied from the first task and the final few layers of the network (shown in blue) are then retrained on a new task such as skin lesion classification for which training data is more scarce. Source: [21].

2.4 Deep Neural Networks

Deep Neural Networks (DNNs) refer to ANNs with multiple hidden layers.

The core concept lies in the hierarchy of layers, used to transform input data into a progressively more abstract and composite representation, thus achieving impressive performances across various domains such as computer vision, natural language processing, and speech recognition.

There is a wide range of architectures for DNNs. For instance:

- Feedforward Neural Networks (FNNs): The simplest form of DNNs, where information flows in one direction from input to output.
- Convolutional Neural Networks (CNNs): Specialized for processing grid-like data (e.g., images), CNNs use convolutional layers to detect spatial hierarchies of features.
- Recurrent Neural Networks (RNNs): Designed for sequential data (e.g., time series, text), RNNs incorporate loops to retain memory of previous inputs, making them suitable for tasks like language modeling and speech recognition.
- Transformers: A more recent architecture relying on self-attention mechanisms, transformers have revolutionized natural language processing (e.g., BERT, GPT).

The depth of these networks allows them to learn hierarchical representations; lower layers detect simple features (edges, textures), while higher layers combine them into complex structures (objects, sentences).

2.4.1 CNNs

Convolutional Neural Networks (CNNs) represent an architectural approach, which can be viewed as a sparsely connected multilayer network with parameter sharing, exploiting invariances and equivariances specific to image input data.

More specifically, an image consists of a rectangular array of pixels, either gray-scale or, more commonly, having a Red-Blue-Green (RGB) value, depending on

the intensity level of each color. Image data can be characterized by their high dimensionality and considering them as unstructured can result in requiring an infeasible amount of parameters to be trained for a network.

CNNs leverage the structure of images as a data type. The pixels of an image have a well-defined spatial relationship with each other, with an image often being represented as a two dimensional array. In natural images, nearby pixels have usually similar colors and intensities, thus having highly correlated values. This provides useful prior knowledge, leading to models with fewer parameters and strong generalization abilities.

In practice, employing the two-dimensional structure of the image data is based on several important concepts outlined below.

Hierarchy

There is an inherent **hierarchical** structure defining images, and hierarchical models fit naturally within the DL framework, which already allows very complex concepts of the input data to be extracted through the many layers of a network.

For instance, in the common task of face detection, images contain faces, which in turn contain eyes, and eyes can be decomposed in the lids, the iris, etc. At the lowest level of the hierarchy, a node of a network can detect simple features, such as edges, with progressively higher levels capturing more complex patterns, such as shapes, facial components, and ultimately, the entire face.

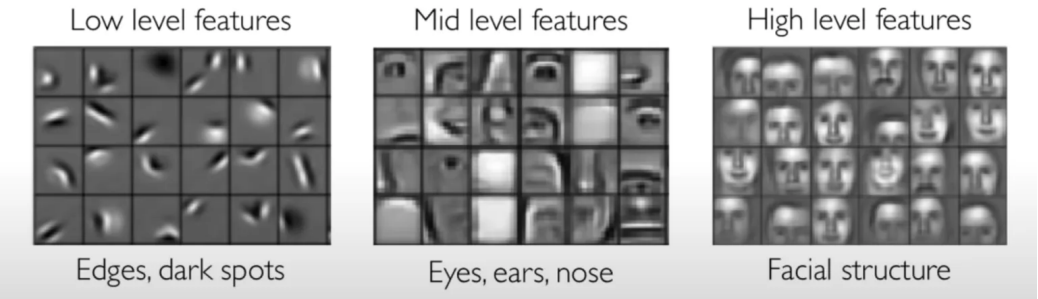


Figure 2.8: Hierarchical representation in deep learning. Source: Intro to Deep Learning.

Locality

In CNNs, each unit processes a small patch of the image so that the network can focus on local patterns and spatial features, allowing it to learn complex representations hierarchically, as previously mentioned, from simple ones.

These patches are referred to as the receptive field of that unit and capture the notion of **locality**. Optimally, the weights associated with them detect some useful low-level feature of the image.

This grid formed by the weights is defined as filter, or kernel, and can also be visualized as an image. If we consider the weights of a unit as fixed, we can mathematically show that maximum activation occurs when the input patch closely resembles the kernel, therefore making the unit act as a **feature detector** that activates when the patch sufficiently matches the kernel.

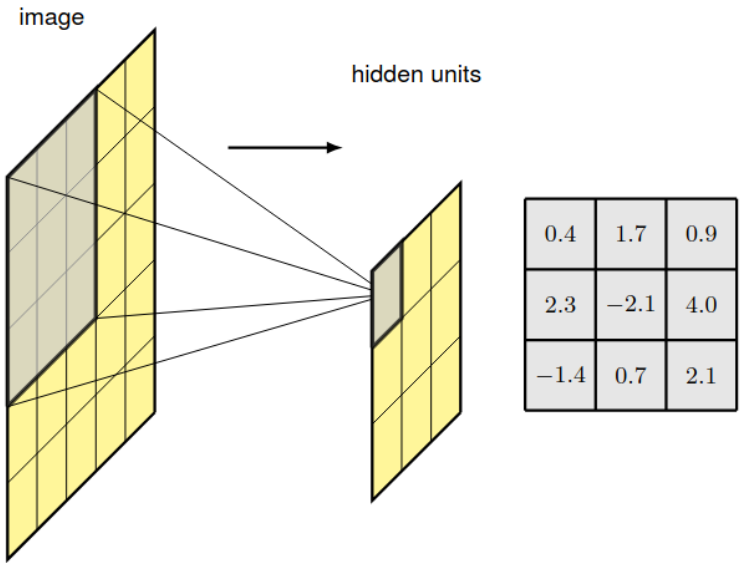


Figure 2.9: (a) Illustration of a receptive field, showing a unit in a hidden layer of a network that receives input from pixels in a 3 × 3 patch of the image. Pixels in this patch form the receptive field for this unit. (b) The weight values associated with this hidden unit can be visualized as a small 3 × 3 matrix, known as a kernel. There is also an additional bias parameter that is not shown here. Source: [21].

Translation equivariance

In mathematics, equivariance is a form of symmetry for functions from one space with symmetry to another.

Similarly, **equivariance** in CNNs refers to the property where a transformation applied to the input results in a corresponding transformation in the output.

Specifically, standard CNNs exhibit translation equivariance, meaning that shifting the input image leads to a proportional shift in the output feature maps, rather than an entirely new output.

This property arises from the use of **convolutional** layers, which apply the same filters across different spatial locations and is beneficial because it allows CNNs to detect patterns regardless of their position in the input.

Figure 2.10: Illustration of edge detection using convolutional filters showing (a) the original image, (b) the result of convolving with a filter that detects vertical edges, and (c) the result of convolving with a filter that detects horizontal edges. Source: [21]

The convolutional structure results in many advantages:

1. Far fewer weights, due to them being shared.
2. The same network can be reused for different-sized input images.
3. Exploit parallelism of Graphic Processing Units (GPUs), leading to efficient implementations.

Padding

Padding refers to the process of adding pixels around an image (or a feature map) to change its shape. It is a common practice in CNN development in case the

resulting feature map convolved with the - usually square shaped - filters has to be adjusted to match the original image. The intensity values of the padded pixels are typically zero, after initially normalizing the rest of the pixels by subtracting their average intensity value.

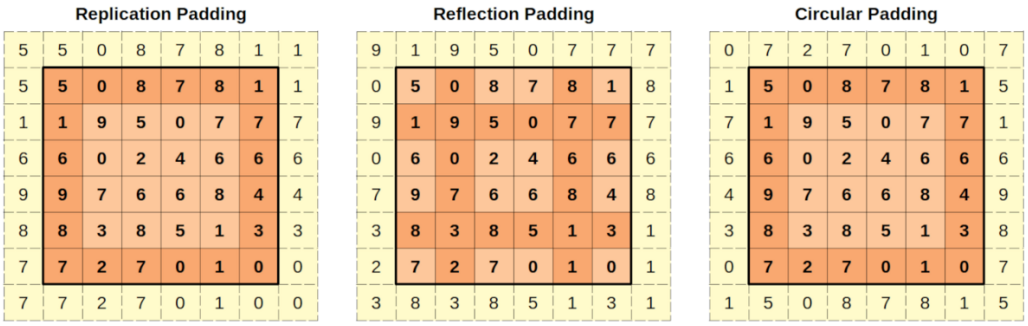


Figure 2.11: Three example padding conditions. Replication condition means that the pixel outside is padded with the closest pixel inside. The reflection padding is where the pixel outside is padded with the pixel inside, reflected across the boundary of the image. The circular padding is where the pixel outside wraps around to the other side of the image. Source: Wikimedia Commons.

Strided Convolutions

As previously mentioned, convolution involves two components: the image and the filter. Since input images are typically much larger than the filters, **strided convolutions** are used to efficiently process the data. In this approach, the filter slides over the image like a window, but instead of moving one pixel at a time, it advances by a specified number of pixels - this step size is known as the **stride**.

Pooling

Pooling is a key operation in CNNs that introduces a degree of translation **invariance** and helps reduce the spatial dimensions of feature maps.

As previously mentioned, convolutional layers ensure translation equivariance. Many tasks however, benefit from invariance to small translations (e.g., image classification). Pooling serves this purpose by *summarizing* local regions in a fixed way, such as taking the maximum or average of values in a region. This helps retain

information about the presence and strength of features while ignoring precise positional details.

This results not only in compressing the data, but also encourages the network to focus on the most relevant features. Furthermore, pooling contributes to the flexibility of CNNs when handling inputs of varying sizes. By adjusting the stride or pooling strategy, the network can maintain a consistent output size, which is especially important for the fully connected layers. [21]

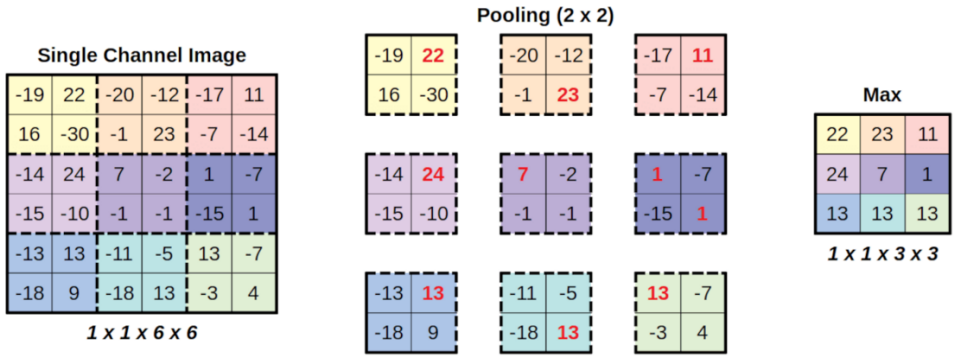


Figure 2.12: Max Pooling is commonly used in CNNs to reduce the spatial dimensions of feature maps. This image presents a worked example of max pooling, with filter size $f = 2$ and stride $s = 2$. Source: Wikimedia Commons.

2.4.2 Transformer

The transformer architecture was proposed in 2017 in the paper "*Attention Is All You Need*" [25] and is considered a major milestone in DL architectures, providing performance improvements in natural language processing, computer vision, and other domains.

Transformers often outperform CNN and RNN models, since they are based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.

As implied by the term, the transformer architecture essentially transforms a set of data in a vector space into another vector space, with data encapsulating a richer representation of the individual inputs.

The transformer architecture operates through stacked layers, each consisting of a multi-head self-attention mechanism and a feedforward network. The input tokens

are embedded and enriched with positional encodings, processed by each layer to extract increasingly abstract representations, ultimately producing an output sequence with richer semantic meaning.

Encoder and Decoder

The original transformer model is composed of two main components: an **encoder** and a **decoder**. The encoder processes the input sequence and encodes it into a set of continuous representations. The decoder then uses these representations, along with its own input (typically the previously generated tokens in tasks like translation), to generate an output sequence.

Each encoder layer consists of a multi-head self-attention mechanism followed by a position-wise feedforward network, with residual connections and layer normalization. The decoder layers are similar but include an additional attention mechanism that attends to the output of the encoder, allowing the decoder to focus on relevant parts of the input sequence. This structure enables the model to effectively handle sequence-to-sequence tasks such as machine translation, summarization, and more.

Attention

The core concept around transformers is *attention*, which - unlike standard ANNs - allows them to assign different weights to different inputs, with coefficients that themselves are dependent on the input data, capturing inductive biases related to not only sequential, but unordered input as well.

The set of input tokens is transformed into output tokens via multiplication with attention weights, which grow proportionally to the influence of an input token on the output.

In the core attention mechanism of the transformer, each input token is projected into three distinct vector representations:

- **Q** (Query): Represents the current element seeking context.
- **K** (Key): Encodes characteristics of all elements in the sequence to be compared against the query.

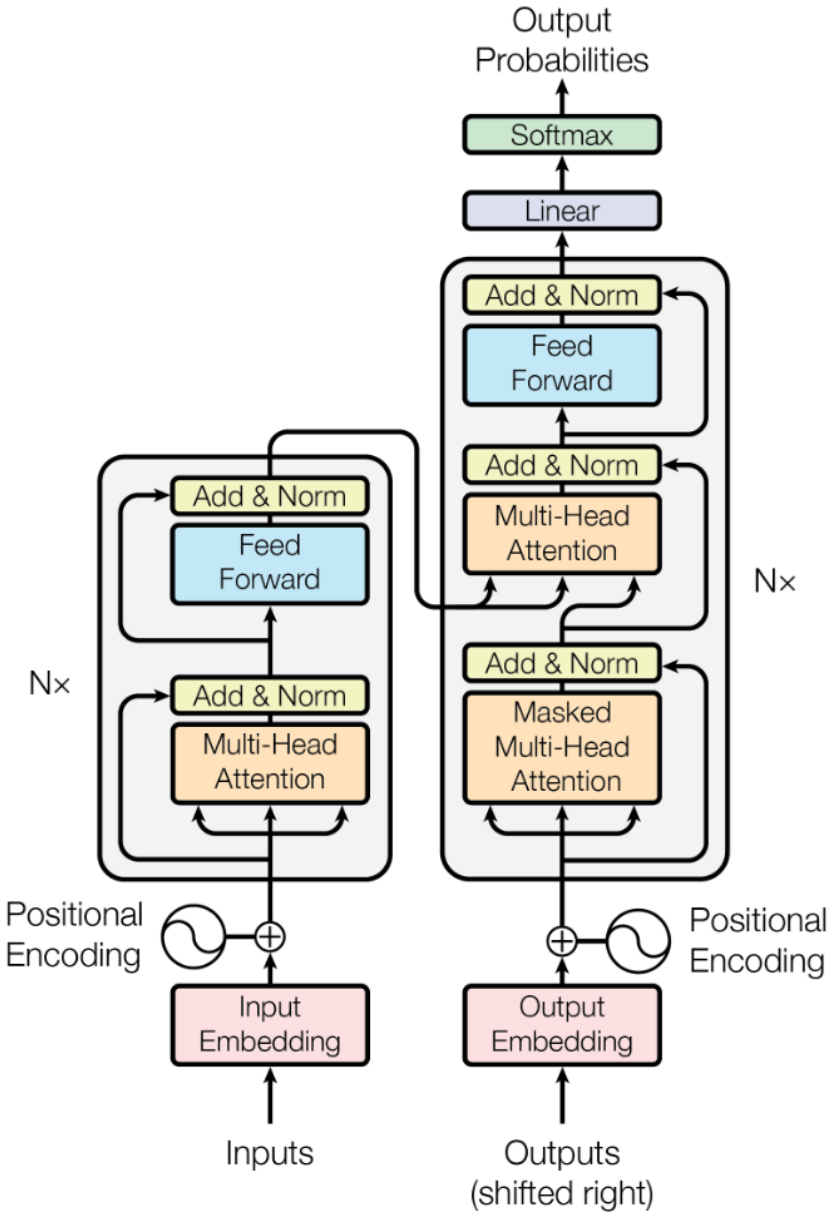


Figure 2.13: The Transformer - model architecture. Source: [25].

- **V** (Value): Contains the information to be aggregated or attended to.

These projections are computed through learned weight matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V$$

where $\mathbf{X}$ is the matrix of input tokens, and $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$ are learned parameter matrices.

The attention mechanism then computes a score based on the compatibility of queries and keys, typically via scaled dot-product attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

Transformers typically employ multi-head attention, where multiple attention mechanisms run in parallel, allowing the model to capture diverse types of relationships between tokens.

Positional Encoding

The transformer architecture lacks an inherent mechanism to process data in a sequential or spatial order. To compensate for this, **positional encoding** is introduced to encapsulate information about the relative or absolute position of input elements into the model.

Typically, positional encoding vectors are added to the input embeddings at the lowest layer of the network. These vectors have the same dimensionality as the input embeddings and are typically generated using deterministic functions such as sine and cosine waves of varying frequencies.

Advantages over other architectures

Transformers have become dominant primarily due to the following advantages over previous architectures:

- Effective on **transfer learning**, usually pre-trained on large amount of data serving as a *foundation model*.
- Ever-improving due to the *scaling hypothesis*, which supports increasing the learning parameters as well as the training data, results in enhanced performance.
- Well-suited for parallel computation on processing units such as GPUs, enabling the training of models with trillions of parameters.

This architecture has since become a key component in many models across various domains. [21]

2.5 Foundation Models

In the paper *"On the Opportunities and Risks of Foundation Models"*, by Rishi Bommasani, Drew A. Hudson, et al., (2021), the authors state that models trained on a wide range of large-scale data are called foundation models to highlight their central but incomplete character.

Foundation models have grown mainly in the subfield of NLP, much as DL was popularized in Computer Vision. We understand these models as a general paradigm of AI, rather than specific to NLP in any way.

Foundation models are possible through transfer learning, but the actual powerful capabilities of them derive from vast amount of data. The importance of the scale and availability of data, as well as the ability to exploit it cannot be underlined enough. Transfer learning with annotated datasets has been common practice for at least a decade, but the non-trivial cost of annotation imposes a practical limit on the benefits of pretraining.

In practice, a foundation model is a large-scale model that can subsequently be adapted to solve multiple different tasks. [26]

Early examples of foundation models are language models (LMs) like OpenAI's GPT series and Google's BERT. Beyond text, foundation models have been developed across a range of modalities-including DALL-E for images, MusicGen for music, and RT-2 for robotic control.

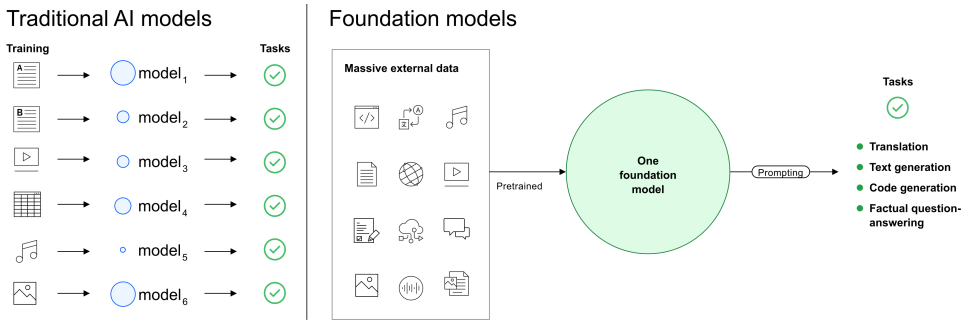


Figure 2.14: As shown in the diagram, traditional AI models specialize in specific tasks. Most traditional AI models are built by using ML, which requires a large, structured, well-labeled data set that encompasses a specific task that you want to tackle. In contrast, foundation models are trained on large, diverse, unlabeled datasets and can be used for many different tasks. Source: IBM watsonx.

2.6 Image Segmentation

Computer Vision is a branch of Artificial Intelligence focused on developing methods to process, analyze, and understand digital images, resulting in the extraction of meaningful information from them.

One of the most popular fields of Computer Vision is Image Segmentation. Image Segmentation refers to the process of dividing an image into meaningful and non-overlapping regions of interest. The development of this field is related to many advancements, e.g., intelligent medical technology, self-driving vehicles, augmented reality, and image search engines.

2.6.1 Taxonomy of tasks

There are three main tasks in Image Segmentation:

1. *Semantic Segmentation*: Labels each pixel with a class, ignoring instance distinctions.
2. *Instance Segmentation*: Labels each pixel with both class and instance identity.

3. *Panoptic Segmentation*: Assigns every pixel a class and a unique instance identity.

Figure 2.15: Semantic, panoptic, and instance segmentation. Source: Youssef's Substack.

2.6.2 Classic approaches

The classic segmentation algorithms were initially proposed for grayscale images, which depend on gray-level similarity in the same region and discontinuity in different ones.

Edge detection is based on gray-level discontinuity, in which the positions where the gray level changes sharply in an image are generally the boundaries of different regions. The ultimate goal is to identify these boundaries or edges. Irwin Sobel and Gary M. Feldman presented the idea of an *"Isotropic 3 × 3 Image Gradient Operator"* in 1968, also known as Sobel operator. It is technically a discrete differentiation operator, computing an estimation of the gradient of the image intensity function, essentially creating an image emphasising edges. [27] Another common technique utilizes the Laplace operator, a differential operator given by the divergence of the gradient of a scalar function. It used in Image Processing and Computer Vision, specifically in tasks such as blob and edge detection. Finally, the Canny edge detector, originally proposed by John F. Canny in 1986, prevails as one of the most successful implementations.

Region division uses gray-level similarity and can be categorized into parallel and serial approaches. A common parallel method is thresholding, where a threshold value - often chosen from the valleys in a grayscale histogram - is used to separate regions. The most suitable threshold is typically found using the zeroth- or

first-order cumulant moments of the histogram, aiming to maximize the distinction between different regions. Another notable technique is the watershed algorithm, which treats the grayscale image as a topographic surface and finds region boundaries by detecting where water would naturally collect and divide. On the other hand, the serial approach breaks down the segmentation process into a sequence of steps, typically involving techniques like region growing and region merging, which are executed one after the other.

Clustering-based methods, such as K-means, were also applied early on by grouping pixels into clusters based on similarity in gray-level or color features, providing an intuitive segmentation strategy. They can be considered a special thresholding segmentation algorithm based on the Lloyd algorithm, originally proposed by Stuart P. Lloyd at Bell Labs in 1957.

2.6.3 Modern approaches

Before Deep Learning was applied to the field of Image Segmentation, semantic texton forests [28] and random forest [29] techniques were generally used to implement segmentation classifiers. In recent years, Deep Learning algorithms have been slowly established and they perform significantly better than the traditional ones.

In 2015, **Fully Convolutional Networks (FCNs)** [30] were proposed, which - unlike standard CNNs - transform the height and width of intermediate feature maps to match those of the input image, thus enabling dense, pixel-wise classification for tasks such as semantic segmentation.

The **encoder-decoder architecture** is a foundational approach for semantic segmentation, building on FCNs. While traditional CNNs like LeNet-5, AlexNet, and VGG are effective for image classification, segmentation requires restoring high-level feature maps to the original image resolution. The encoder extracts semantic features through convolution and pooling operations using backbones such as VGG, Inception, and ResNet. The decoder then upsamples these features to produce pixel-level class predictions (with methods such as interpolation, transposed convolution, unpooling, etc.).

Skip connections were developed to tackle the degradation problem and improve rough pixel positioning. They help the network remember the original shapes and edges, so the final segmentation mask can accurately follow the boundaries of objects, not just roughly guess them. The U-Net architecture [31] proposed a new

long skip connection, in order to obtain fine-grained details of images, and is now widely used in research on medical image segmentation.

DeepLab and its various versions (e.g., DeepLabv3+) [32] introduced innovations such as *atrous (dilated) convolutions* to solve the problem of resolution reduction during up-sampling. This allowed the network to understand both fine object boundaries and larger semantic context. [33]

In 2020, the paper "*An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*" was published and proposed a **Vision Transformer (ViT)**, proving that transformers could substitute for CNN in classification and prediction of image patch sequences. They divided the image into equally-sized patches, lined them up and fed the patches sequence vector into a transformer encoder that consisted of alternating multi-head attention layers and multi-layer perceptron (MLP) (Figure 2.16). [34]

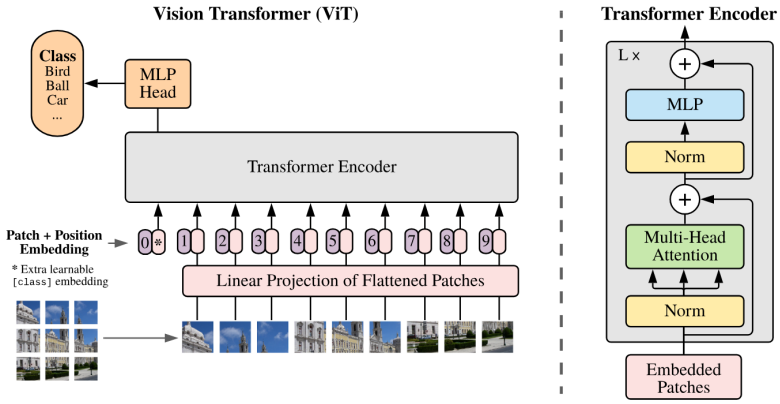


Figure 2.16: The ViT architecture. Source: [34].

More recently, in 2023, the paper *"Segment Anything"* was published by the META AI Research Team. Alongside the paper, the project included a new task, model and the largest dataset for image segmentation. The Segment Anything Model (SAM) is based on foundation models that have significantly impacted natural language processing (NLP) and focuses on promptable segmentation tasks. [35] One year later, in 2024, the research team released *"SAM 2: Segment Anything in Images and Videos"*, which is based on a pure transformer applied directly to sequences of image patches, discarding CNNs. [36]

Figure 2.17: The Segment Anything Model produces high quality object masks from input prompts such as points or boxes, and it can be used to generate masks for all objects in an image. It has been trained on a dataset of 11 million images and 1.1 billion masks, and has strong zero-shot performance on a variety of segmentation tasks. Source: Segment Anything, GitHub.

2.6.4 Watershed Algorithm

In the field of Image Processing, a *watershed* is a transformation defined on a grayscale image. The paper *"Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations"*, (IEEE PAMI, 1991), by Luc Vincent and Pierre Soille, revolutionized the execution with an efficient immersion-based algorithm along with the pseudocode. [37]

The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges. In practice, the watershed is a classical algorithm used for segmentation, that is, for separating different objects in an image.

2.6.5 Evaluation Metrics

In image segmentation tasks, the performance of a model is commonly evaluated using metrics that quantify the overlap between the predicted segmentation mask

and the ground truth. Two widely used metrics for this purpose are the Intersection over Union (IoU) and the Dice Coefficient.

The **IoU**, also known as the Jaccard Index, measures the ratio of the intersection to the union of the predicted mask P and the ground truth mask G, and is defined as:

$$\text{IoU}(P, G) = \frac{|P \cap G|}{|P \cup G|} \tag{2.1}$$

On the other hand, the **Dice Coefficient**, also referred to as the Dice Score, emphasizes the similarity between the predicted and ground truth masks by calculating twice the intersection over the total number of pixels in both masks:

$$\text{Dice}(P, G) = \frac{2|P \cap G|}{|P| + |G|} \tag{2.2}$$

While both metrics range from 0 to 1 (with 1 indicating perfect segmentation), the Dice Coefficient is often more sensitive to small objects and imbalanced classes, making it particularly suitable for medical and fine-grained segmentation tasks.

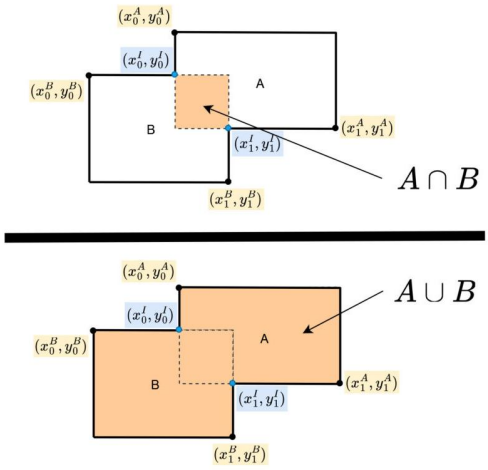


Figure 2.18: Illustration of IoU calculation. Source: Machine Learning Space.

2.7 SAM 2

As previously mentioned, the Segment Anything Model (SAM) is a foundation vision model developed by Meta AI for general-purpose image segmentation. It supports zero-shot segmentation using user-provided prompts such as points, boxes, or masks. SAM 2 is an improved version that enhances segmentation quality and robustness, particularly in challenging or fine-grained scenarios, while also supporting video segmentation.

Their work include a task, a model and a dataset. In this section, we analyze SAM 2 with a focus on its image segmentation capabilities, as our task involves static mosaic images, thereby setting aside details related to video segmentation.

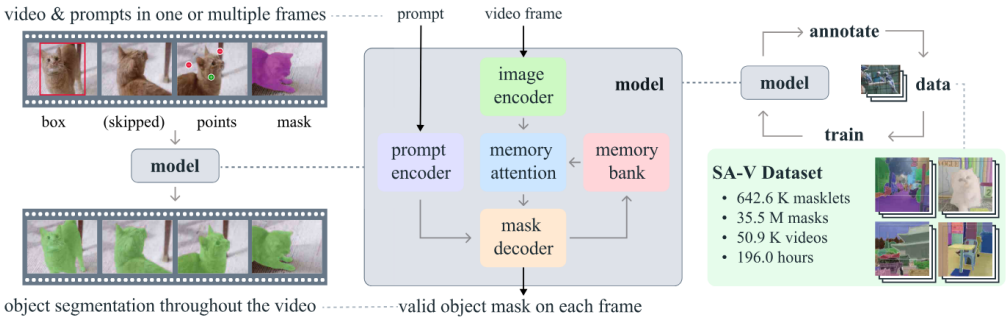


Figure 2.19: Task - Model - Data. Source: [36].

2.7.1 Task

The main task is **promptable visual segmentation**. SAM 2 supports prompts by clicks, boxes or masks, and they can be positive or negative to include or exclude a region from the final segmentation.

2.7.2 Model

SAM 2 can be seen as a generalization of SAM, retaining its core functionality for image segmentation while introducing architectural enhancements.

The prompt encoder handles sparse inputs via positional encodings and learned

embeddings, while masks are embedded using convolutions and added to the frame embedding.

The mask decoder employs ”two-way” transformer blocks to iteratively refine both prompt and image embeddings. For ambiguous prompts, SAM 2 outputs multiple segmentation hypotheses, selecting the one with the highest predicted Intersection over Union (IoU).

Furthermore, SAM 2 incorporates hierarchical image encoding via a pre-trained Hiera backbone[1], enabling the use of multiscale features and skip connections during decoding. These refinements improve segmentation quality and efficiency, particularly in complex or high-resolution image tasks.

The memory components of SAM 2, including the memory encoder and memory bank, are specifically designed to capture temporal information across video frames and are therefore not utilized in this work.

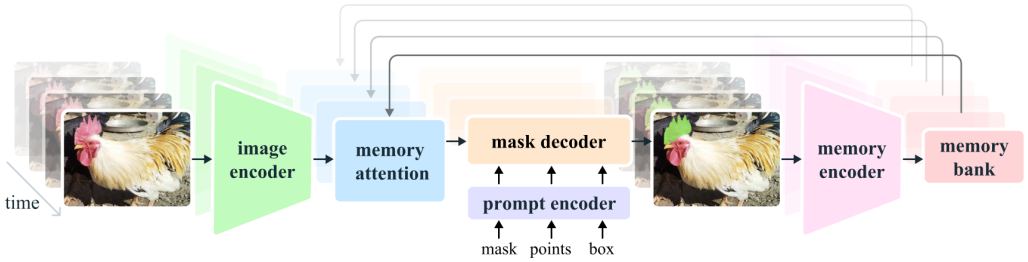


Figure 2.20: The SAM 2 architecture. Source: [36].

2.7.3 Data

SAM 2 is trained jointly on a diverse combination of large-scale image and video datasets to support its promptable visual segmentation capabilities across both domains. The image datasets provide richly annotated objects to enable the model to learn generalizable segmentation from sparse prompts such as points, boxes, and masks. Video datasets are used to teach temporal consistency and object tracking through memory mechanisms.

During training, interactive prompting is simulated by randomly selecting frames and applying prompts such as clicks, bounding boxes, or masks to guide segmenta-

[1]The Hiera backbone is a hierarchical Vision Transformer (ViT) that extracts multi-scale features to better capture both local details and global context for segmentation tasks.

tion refinement. This approach enhances the model's ability to iteratively improve segmentation masks based on user input. [36]

2.7.4 Image Predictor and Automatic Mask Generator

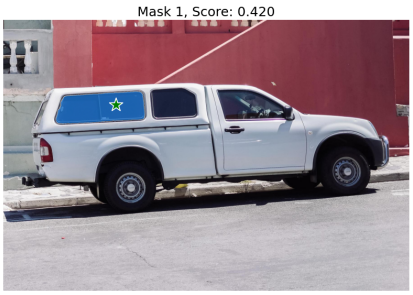
The researchers provide two main pathways for image segmentation of static image use cases depending on whether the prompts are created manually or automatically.

SAM 2 predicts object masks given prompts that indicate the desired object. The model first converts the image into an image embedding that allows high quality masks to be efficiently produced from a prompt.

The SAM 2 *Image Predictor* provides an easy interface for image prompting and allows the user to first set an image and in turn calculates the necessary image embeddings. Then, prompts can be provided to efficiently predict masks. The model can take as input both point and box prompts, as well as masks from the previous iteration of prediction. This manual method is suitable for training the model, where we are supervising the process, but there is the need for an automatic pipeline for our specific task.

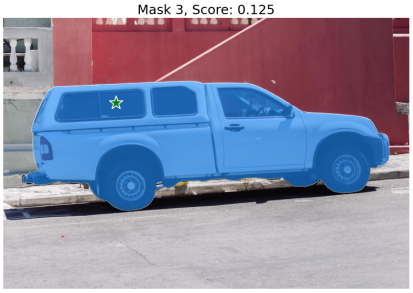
Since SAM 2 can efficiently process prompts, masks for the entire image can be generated by sampling a large number of prompts over an image. The SAM 2 *Automatic Mask Generator* implements this capability. It works by sampling single-point input prompts in a grid over the image, from which the model can predict multiple masks. Then, masks are filtered for quality and deduplicated using Non-Maximum Suppression[2]. Additional options allow for further improvement of mask quality and quantity, such as running prediction on multiple crops of the image or postprocessing masks to remove small disconnected regions and holes.

[2]Non-Maximum Suppression (NMS) is a method used in object detection to remove extra boxes that are detected around the same object. When an object is detected multiple times with different bounding boxes, NMS keeps the best one and removes the rest. This helps to make sure each object is counted only once, improving the accuracy and clarity of the results.



(a) Mask 1 with model score: 0.420.

(b) Mask 2 with model score: 0.402.



(c) Mask 3 with model score: 0.125.

Figure 2.21: Example of multimask output from the SAM 2 Image Predictor. A single input point (visible as a star in each mask) guides the model to segment the object of interest. For ambiguous prompts, the model outputs multiple segmentation masks, each with an associated quality score. Source: SAM 2 Image Predictor Example.

(a) Example input image with many possible layers and regions of interest.

(b) Generated masks using a configuration tuned to capture only the most prominent regions.

(c) Generated masks using a configuration that captures more fine details and produces more masks.

Figure 2.22: Example outputs from the SAM 2 Automatic Mask Generator. The generator provides several tunable parameters controlling how densely points are sampled and what thresholds are applied to filter out low-quality or duplicate masks. Additionally, automatic cropping can be used to improve performance on smaller objects, and post-processing steps can refine masks by removing stray pixels and filling holes. Source: SAM 2 Automatic Mask Generator Example.

Proposed Method

3.1 Motivation for using SAM 2

We have established the necessary theoretical background to understant the basic mechanisms of the SAM 2 model. In this section we are discussing the motivation for selecting this model.

The SAM 2 model is a foundation model, which was exposed to millions of training images and masks. The sheer amount of data used by the META AI researchers resulted in a model with very powerful segmentation capabilities and high accuracy, while also creating a component that can be fine-tuned enabling it to specialise and personalise, adapting to domain-specific tasks, user preferences, and real-world constraints without full retraining. This is very important as it creates a strong foundation upon which anyone can build by *transfer learning*, a technique already discussed in previous sections. [38]

SAM 2 is well-suited for our task mainly due to its ability to perform general-purpose segmentation from sparse prompts makes it adaptable to new domains with limited labeled data. [39] As a foundation model, SAM 2 captures a rich representation of visual structures, allowing us to leverage these capabilities through transfer learning. [40] [41] [42]

Additionally, SAM 2 exhibits strong zero-shot segmentation capabilities, which further supports its application to novel and diverse datasets without requiring extensive retraining [43].

3.2 Data Acquisition

The first step in addressing the problem of mosaic image segmentation is data acquisition. While many Computer Vision tasks benefit from publicly available datasets, niche applications - such as our task - often require manual annotation. We retrieved our images from various public sources (e.g., iStock by Getty Images, Adobe Stock) and not from a unified dataset, thus constructing our own dataset.

The researchers of SAM 2 have provided specific instructions about the input data regarding the process of fine-tuning their model.

The annotation process requires a considerable amount of time because of the nature of the problem; most tesserae have complex shapes and appear damaged. To tackle this difficulty we are employing image augmentation techniques to further expand the annotated dataset.

3.2.1 Data Annotation

The data annotation process includes retrieving mosaic images from public sources and producing the regions of interest, namely the tesserae, which will be referred to as *masks*. These masks are aggregated into a different image, which serves as the ground truth image.

Based on the instructions of the official GitHub page, the masks should be black and the background white, resulting in a *binary* ground truth image. Additionally, every mask is expected to be accompanied by an input point, since SAM 2 is promptable.

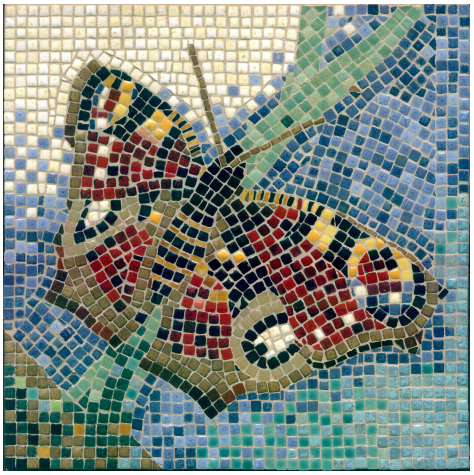
To perform the annotation we operate with Label Studio, which is an open source Data Labeling platform. In this way the process becomes easier and the annotations more accurate. The input points are created automatically by computing the centroid of each annotated region.

3.2.2 Data Augmentation

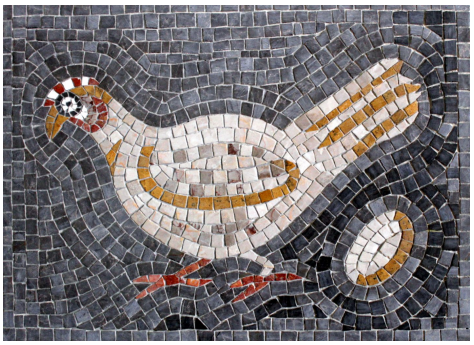
To address the limited size of the dataset and enhance model generalization, data augmentation is applied to a subset of the annotated images used for training and validation. This process aims to artificially increase data diversity, reduce overfitting, and help the model become more robust to variations in tesserae appearance and mosaic structures.

Specifically, the images are first systematically cropped and then augmented through rotations of $90°$, $180°$, and $270°$, as well as vertical and horizontal flips. These transformations preserve the semantic content of the data while introducing variations in the orientation, which is vital as the tesserae can appear at arbitrary angles.

This augmentation pipeline not only expands the effective size of the dataset but also encourages the model to learn rotational and reflection invariance.



(a) Butterfly (1575×1574)



(b) Chicken (1425×1024)

(c) Duck (1200×1021)

(d) Duck modern (2542×2560)

(e) Horse (966×710)

(f) Floor Pattern (5) (2048×836)

Figure 3.1: Original mosaic images (part 1 of 2).

(a) Floor Pattern (1) (500×402)

(b) Floor Pattern (2) (2048×1536)

(c) Rabbit (894×894)

(d) Tiger (1365×1024)

(e) Floor Pattern (6) (2048×834)

(f) Lizard (894×894)

Figure 3.2: Original mosaic images (part 2 of 2).

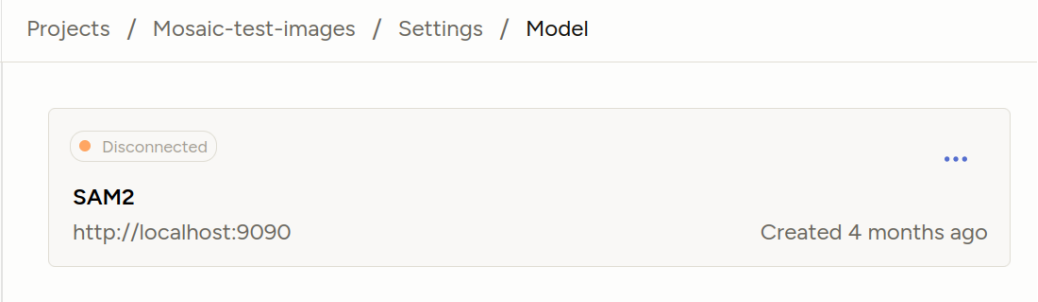


Figure 3.3: The Label Studio platform supports automatic segmentation in order to accelerate the manual annotation. We configured the SAM 2 model locally in order to use it on the platform.

3.2.3 Training and Validation Images

As previously mentioned, the training and validation images must be mapped to their respective ground truth images.

For every mask in a ground truth image, an input point must be given inside the region of the mask, as SAM 2 is promptable and uses these points to make predictions, which subsequently are compared to the ground truth images.

Based on the similarity of the ground truth image with the model's predictions, the internal parameters are fine-tuned.

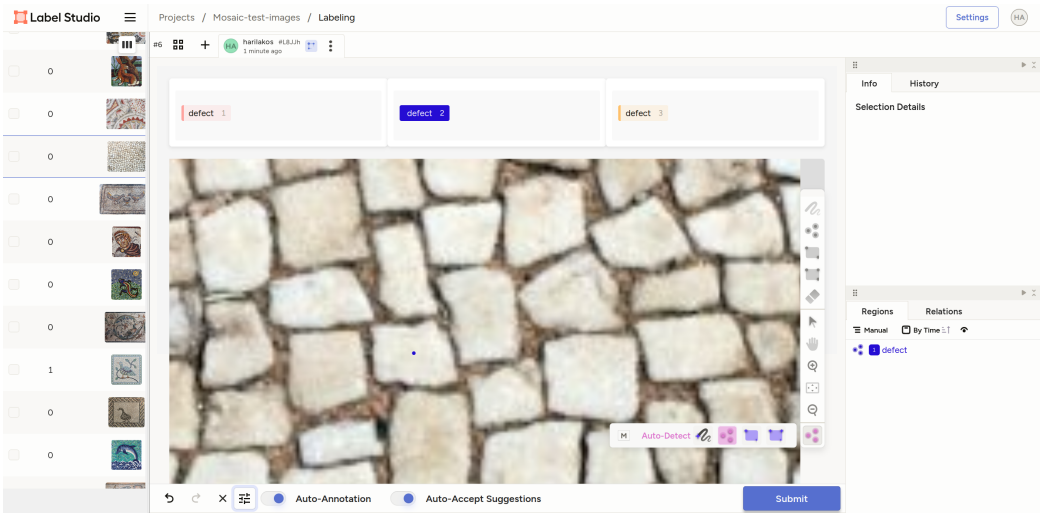
In this implementation, the centroid of every mask region is computed and a point is assigned with the corresponding coordinates.

Another requirement for the input images is that their size must not exceed 1024 pixels; therefore, both the original and ground truth images are resized accordingly.

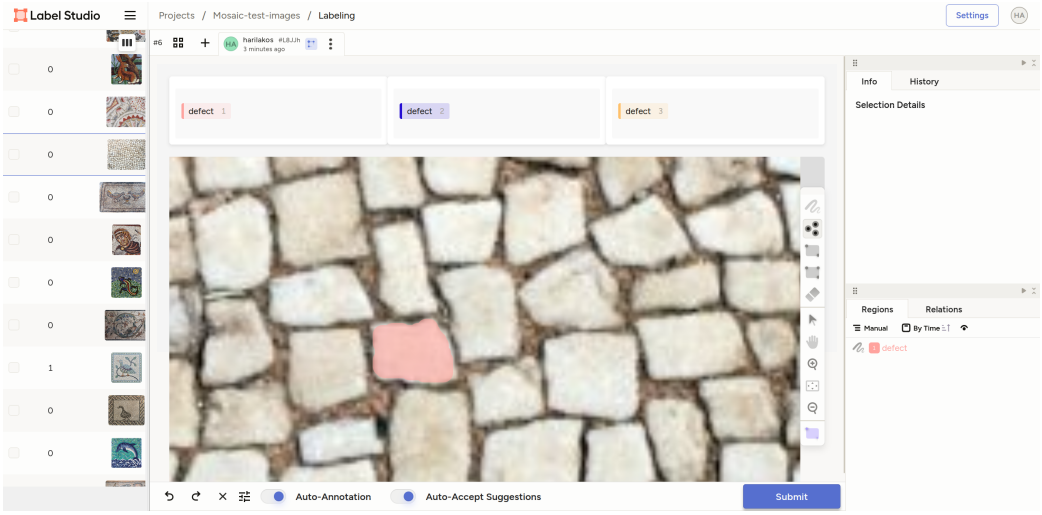
3.3 Fine-tuning the SAM 2 Model

3.3.1 Pre-trained Models

Along with the paper and code, the researchers of the SAM 2 project have provided a list of pre-trained models and their configurations. As shown in Table 3.1, the performance increases with the model size at the cost of speed.

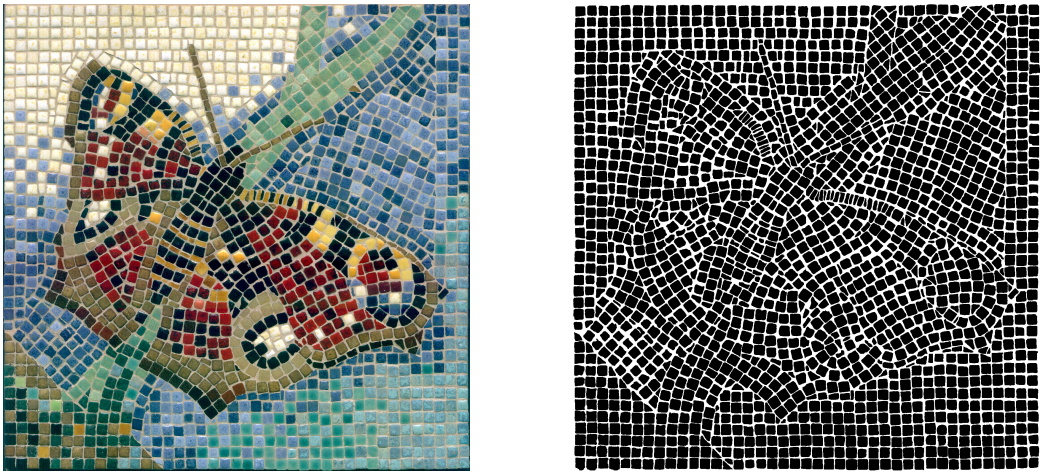


(a) In Label Studio we can set a point inside the region of our interest, e.g., a tessera, and automatically detect the mask.



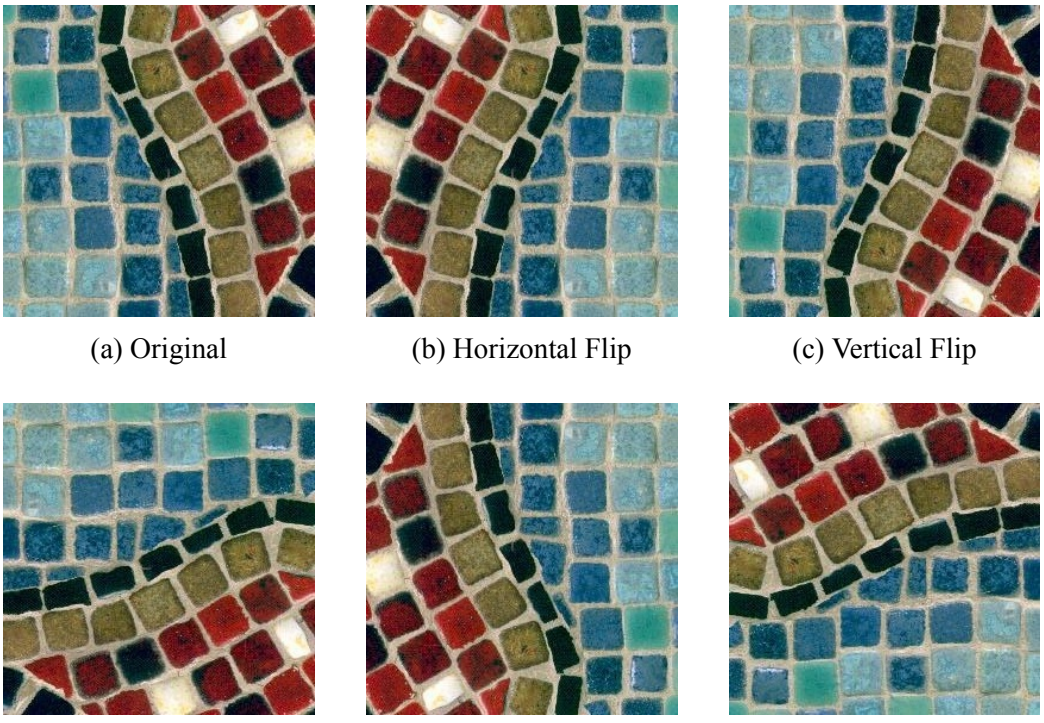
(b) This is the mask produced automatically by the point above.

Figure 3.4: Example of the iterative process followed in the Label Studio platform: we set a point for SAM 2 to detect the tessera that it belongs to and the model automatically segments based on the point.



(a) Original mosaic image (b) Corresponding binary mask

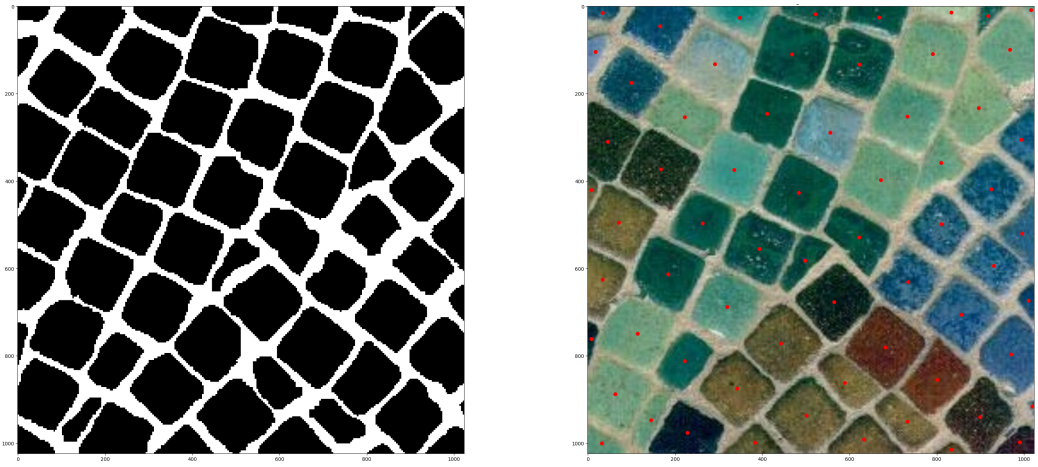
Figure 3.5: Example of a mosaic image and its binary mask.



(a) Original (b) Horizontal Flip (c) Vertical Flip

(d) Rotation 90° (e) Rotation 180° (f) Rotation 270°

Figure 3.6: Augmentations applied to a single example image (cropped).



(a) Ground Truth image (b) Input Points

Figure 3.7: Example of input points.

Model	Size (M)	Speed (FPS)	MOSE val (J&F)
`sam2_hiera_tiny`	38.9	91.5	70.9
`sam2_hiera_small`	46.0	85.6	71.5
`sam2_hiera_base_plus`	80.8	64.8	72.8
`sam2_hiera_large`	224.4	39.7	74.6

Table 3.1: Performance of SAM 2 Hiera models on the MOSE validation set. Size is given in millions of parameters (M) and speed in frames per second (FPS). Source: facebookresearch/sam2.

In this work, we fine-tune the `sam2_hiera_large` model, which offers the best segmentation performance among the available options, despite its larger size and slower inference speed.

3.3.2 Training Components

The SAM 2 architecture consists of three basic components. The image encoder processes the images and produces the image embeddings. It is the largest component and training it is infeasible with the available computational resources. We choose to train only the prompt encoder and mask decoder, while freezing the image encoder layer.

3.3.3 Training Process

For the training process, we are using the SAM 2 Image Predictor, which is the component that ultimately predicts the tesserae based on the original image and the respective input points.

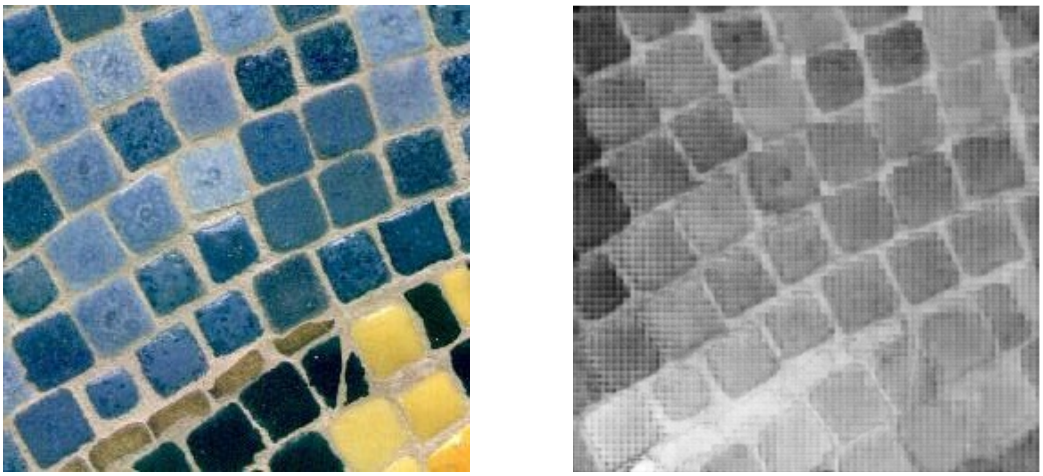
More specifically, the model processes a batch of images, the size of which is a tunable parameter. For an image of the batch the prompt encoder creates sparse and dense embeddings, which in turn are used by the mask decoder in order to construct the predicted masks in low resolution.

Every pixel is assigned to a raw prediction score, also referred to as *logit*, which indicates how confident the model is that it belongs to a tessera (Figure 3.8). The sigmoid function is eventually applied to the logits to transform the real numbers into probabilities.

Loss Function

Loss functions play a vital role in determining the model performance. For sophisticated objectives such as segmentation, it is not possible to decide on a universal loss function. [44]

In the process of supervising the model, in order to enhance its performance for our specific task of segmenting mosaics, we are designing a *loss function* based



(a) Original image (b) Raw mask logits

Figure 3.8: In this example subfigure (a) depicts a random, cropped image of the augmented training dataset, and (b) illustrates the same image with every pixel assigned to a logit, indicating how confident the model is that it belongs to a tessera.

on the Dice Coefficient, which is differentiable, to directly optimize the overlap between the predicted and the ground truth image. [45] [46]

Let P be the predicted mask and G the ground truth mask, both with pixel values normalized between 0 and 1.

The **Dice Loss** is defined as:

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2 \cdot (P \cdot G)}{\|P\|_1 + \|G\|_1}$$

To further improve the model's performance, we also consider the model's confidence in its prediction by comparing the predicted confidence score to the actual overlap between the predicted and ground truth masks. This process is commonly referred to as *confidence calibration*.

Modern neural networks are often overconfident in their incorrect predictions, and SAM 2 exhibits similar behavior. With calibrated confidence, the primary purpose is to find misclassification errors by rejecting low-confidence predictions. [47] [48]

The predicted mask is thresholded at 0.5 to create a binary mask $\hat{P} = 1_{P>0.5}$,

where pixels with predicted values greater than 0.5 are considered foreground.

The Intersection over Union (IoU) between the ground truth G and the thresholded prediction $\hat{P}$ is:

$$\text{IoU} = \frac{|G \cap \hat{P}|}{|G \cup \hat{P}|}$$

The model also outputs a *predicted confidence score* s for the mask, which ideally should reflect this IoU value.

The **Score Loss** penalizes the absolute difference between this predicted confidence score and the actual IoU:

$$\mathcal{L}_{\text{Score}} = |s - \text{IoU}|$$

Finally, the **total loss** function combines the Dice Loss and Score Loss with a weighting factor λ:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{Dice}} + \lambda \cdot \mathcal{L}_{\text{Score}}$$

Based on the value of the loss function, the internal parameters of the model are adjusted through backpropagation and gradient descent.

Validation

The next step involves validating the updated model on a separate validation set, in order to monitor overall performance progression and identify the optimal model checkpoint across training epochs.

While the validation process typically mirrors the training phase in terms of loss computation, in this work we adopt a distinct evaluation strategy due to the specific nature of the task.

In particular, we leverage the Automatic Mask Generator component of SAM 2, which performs mask generation based on a set of predefined parameters, with-

out requiring manual input points. This is important as it simulates the inference conditions where user-provided prompts will not be available.

A grid of points is constructed internally, simulating the inference process where we will not have the input points in advance. For the resulting masks, we are computing a set of metrics, with our attention on the accuracy, IoU and Dice Coefficient.

3.3.4 Hyperparameters

The training process involves tuning of several critical hyperparameters to ensure effective model convergence and optimal performance. In this implementation, we employ the AdamW optimizer. The AdamW optimizer relies primarily on two key hyperparameters: the *learning rate* and the *weight decay coefficient*.

In addition to these optimizer-specific parameters, we also tune the *batch size*, which denotes the number of training samples processed simultaneously in one iteration. The batch size impacts the stability of gradient estimates and computational efficiency; larger batch sizes tend to provide smoother gradient approximations but require greater memory resources, while smaller batch sizes introduce more stochasticity, which can sometimes improve generalization.

The combined tuning of these hyperparameters plays a vital role in the overall training dynamics and model effectiveness, requiring a systematic search to identify an optimal configuration tailored to the specific segmentation task at hand.

Learning Rate

The learning rate controls the extent to which newly learned information updates or replaces previously acquired knowledge. [49]

The learning process heavily depends on choosing a suitable learning rate. A small learning rate could significantly slow the process, while a bigger could mean that the loss function does not converge, thus the learning process fails.

To get the optimal results, a common practice involves using a *learning rate decay*, in which the rate is a monotonically decreasing function of the epoch iteration.

This technique accelerates the learning process while minimizing the risk of the

loss function not converging to a minimum value. [50]

The learning rate η_t at epoch t is defined by the cosine annealing schedule, as introduced by Loshchilov et al. in *"SGDR: Stochastic Gradient Descent with Warm Restarts"* [51]:

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}}) \left(1 + \cos\left(\frac{t\pi}{T_{\text{max}}}\right)\right)$$

where:

- η_{max} is the initial learning rate
- η_{min} is the minimum learning rate
- T_{max} is the total number of epochs
- t is the current epoch

This scheduling strategy gradually reduces the learning rate from its maximum value to the minimum over the course of training, following a cosine function.

The goal is to allow large updates in the early stages for faster convergence, and smaller updates later to fine-tune the model weights.

Weight Decay

Weight decay is a regularization technique to prevent overfitting by penalizing large weights during training.

A penalty term is added to the loss function, which discourages the model from giving too much importance to a single parameter. By penalizing large weights it enhances the generalization ability of the trained model.

In mathematical terms, the total loss function is calculated as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{original}} + \lambda \|\theta\|_2^2$$

where:

- $\mathcal{L}_{\text{original}}$ is the original loss function
- θ represents the model parameters
- λ is the weight decay coefficient

In our implementation, we adopt the decoupled weight decay strategy introduced by Loshchilov and Hutter [22], where weight decay is applied separately from the gradient-based update step. This decoupling, used in optimizers such as AdamW, improves the regularization effect by avoiding interference with adaptive learning rate mechanisms.

Batch Size

The batch size, which is the number of training records used in one forward and backward pass of the network, is a vital parameter of the training process.

Setting this hyperparameter to a high value can make the network have a late convergence, however, if it is too low, it will make the network bounce back and forth without achieving acceptable performance. The nature of the dataset can also influence the batch size parameter. [52]

Therefore, it is important to choose an optimal value in order to balance the computational efficiency and memory constraints, while also maximizing the model generalization.

3.3.5 Training Configurations

In this work, we systematically experimented with various combinations of key adjustable hyperparameters, namely the learning rate, weight decay, and batch size.

By exploring a range of values for each hyperparameter, we aimed to identify an optimal configuration that balances effective learning with model regularization, while also considering computational efficiency.

The exploration was carried out through a series of controlled experiments, enabling us to analyze how each parameter influences the segmentation performance and stability during training.

More specifically, we experiment with the different combinations of the values below:

- Learning Rate: $\{10^{-5}, 2{\cdot}10^{-5}, 3{\cdot}10^{-5}\}$
- Weight Decay: $\{10^{-4}, 10^{-5}\}$
- Batch Size: $\{4, 8\}$

For each run we manually employ *early stopping*, allowing for the training process to complete 10 epochs, retrospectively saving the model with the highest validation Dice Coefficient value.

3.3.6 Optimization Process

The next step in the process is identifying the optimal input parameters of the Automatic Mask Generator for every model and, thus, we are performing Hyperparameter Optimization.

In this work, we are using the Optuna framework to automate the hyperparameter search. We are choosing the *Bayesian Optimization* which requires a set of input variables and an objective function to optimize upon. It is best-suited for optimization over continuous domains of less than 20 dimensions and is based on a Bayesian ML technique, Gaussian process regression, and finally an acquisition function[1]. [53] [54]

We are *optimizing* the value the Dice Coefficient value on a subset of the manually annotated images (2 samples), based on four input parameters of the SAM 2 model, Automatic Mask Generator:

- **Prediction IoU Threshold**: A filtering threshold in [0,1], using the model's predicted mask quality.
- **Stability Score Threshold**: A filtering threshold in [0,1], using the stability of the mask under changes to the cutoff used to binarize the model's mask predictions.

[1] Acquisition functions are mathematical techniques that guide how the parameter space should be explored during Bayesian Optimization.

- **Stability Score Offset**: The amount to shift the cutoff when calculated the stability score.
- **Mask Threshold**: Threshold for binarizing the mask logits.

The objective function used to evaluate the quality of the predicted masks over N input images is defined as:

$$\mathcal{O} = \frac{1}{N} \sum_{i=1}^{N} \text{Dice} \left(M_i^{\text{gt}}, M_i^{\text{pred}} \right) \tag{3.1}$$

where M_i^{gt} and M_i^{pred} denote the ground truth and predicted masks for the i-th input image, respectively. Essentially, it measures the overlap between the ground truth and predicted masks, with higher values indicating better segmentation performance; hence we are *maximizing* this objective.

3.3.7 Testing Configurations

We conducted a series of experiments involving the fine-tuned models derived from the training process.

For comparative purposes, we employ the base versions of SAM 2, specifically the *small* and *large* checkpoints, as reference baselines to contextualize and understand the performance of our fine-tuned models.

The evaluation metrics used to assess model performance include IoU, Accuracy, and Dice Coefficient, each providing complementary insights into the quality of segmentation results.

3.4 Experimental Environment and Dataset Statistics

In this section we describe the experimental environment and the system specifications used for training and evaluation. Additionally, this section includes the specific dataset statistics that we used throughout this work.

3.4.1 Environment

Due to the high computational cost of the SAM 2 model by META AI, we set up our programming environment on the Kaggle[2] platform. Kaggle offers free access to powerful cloud hardware, including GPUs, to support ML development and experimentation.

In our case, we utilized a single **NVIDIA Tesla P100** with **16GiB** of dedicated GPU memory, as provided by the Kaggle runtime. The environment also included up to **29GiB** of **system RAM** and **57.6GiB** of persistent disk space, which proved sufficient for model training and evaluation during our experiments. This setup allowed us to train and evaluate SAM2 under *realistic* hardware constraints.

3.4.2 Datasets

As previously mentioned, we manually annotated mosaic images retrieved from public sources (e.g., iStock by Getty Images, Adobe Stock), constructing a dataset which we split into training and testing sets. Each image consisted of hundreds of tesserae, making it computationally infeasible to process them individually. To reduce computational cost, we consistently cropped the initial images to contain approximately 100 tesserae per image.

All training images were augmented using deterministic geometric transformations (e.g., flips and rotations) to increase generalization. The train dataset consists of 984 augmented mosaic images along with the masks, for which we employed an 80%-20% split between the pure training images and the validation dataset, yielding 787 training and 197 validation images.

The test dataset consisted of 166 cropped mosaic images which amounts to approximately 15% of the initial annotated set.

The resulting proportions are approximately 68% for training, 17% for validation, and 15% for testing. Note that the slightly uneven counts are due to the preprocessing stage: the original mosaic images were cropped into smaller segments. Since the splitting was performed at the image level (not per crop), it was not possible to evenly divide a single mosaic into different sets.

[2]Kaggle is a data science competition platform and online community for Data Scientists and Machine Learning practitioners under Google LLC.

3.5 Runtime Analysis

In this section we are summarizing the runtime details based on the computational power and environment configurations provided by Kaggle, as explained in Section 3.4.

3.5.1 Training Runtime

In this work we trained the SAM 2 Image Predictor by freezing the Image Encoder component and unfreezing the Mask Decoder and Prompt Encoder. We employed the *large checkpoint* of 897.95 MB.

With a total of 984 training and validation augmented images, a single epoch required approximately 20 minutes. For 10 epochs, the total runtime for training was approximately 200 minutes.

Table 3.2: Hyperparameter configurations for Run 7

Run #	Learning Rate	Weight Decay	Batch Size	Epochs
7	10^{-5}	10^{-5}	4	10

3.5.2 Optimization Runtime

In order to find the optimal input parameters for the Automatic Mask Generator, for each model, we conducted an optimization study with the Optuna framework. We used two images as reference to optimize the model.

Each trial required approximately 10 minutes, thus, for 10 trials, the optimization required approximately 100 minutes.

3.5.3 Testing Runtime

We finally evaluated each model on the testing dataset of 166 cropped images. Each model required approximately 60 minutes to finish the evaluation process.

Table 3.3: Summary of Runtime Statistics

	Details	**Runtime**
Training	984 images, 10 epochs	200 minutes (20 min/epoch)
Optimization	10 trials	100 minutes (10 min/trial)
Testing	166 cropped test images	60 minutes (21.7 sec/image)
Total per Model	**Training + Optimization + Testing**	360 minutes or 6 hours

Results

In this chapter we are presenting the results of our research while comparing our fine-tuned model with the baseline checkpoints of SAM 2 by META AI. The training results illustrate the optimal hyperparameter configuration and, alongside the optimization and testing results, it is further proven that our model performs better than the baseline ones.

4.1 Training Results

In this section we explore the training results for the different configurations of our work. We experiment with various combinations of the learning rate, weight decay and batch size.

The Table 4.1 summarizes and enumerates the total runs. This notation is used throughout this work for clarity.

Table 4.1: Hyperparameter configurations for each run

Run #	Learning Rate	Weight Decay	Batch Size
1	10^{-5}	10^{-4}	4
2	$2{\cdot}10^{-5}$	10^{-4}	4
3	$3{\cdot}10^{-5}$	10^{-4}	4
4	10^{-5}	10^{-4}	8
5	$2{\cdot}10^{-5}$	10^{-4}	8
6	$3{\cdot}10^{-5}$	10^{-4}	8
7	10^{-5}	10^{-5}	4
8	$2{\cdot}10^{-5}$	10^{-5}	4
9	$3{\cdot}10^{-5}$	10^{-5}	4
10	10^{-5}	10^{-5}	8
11	$2{\cdot}10^{-5}$	10^{-5}	8
12	$3{\cdot}10^{-5}$	10^{-5}	8

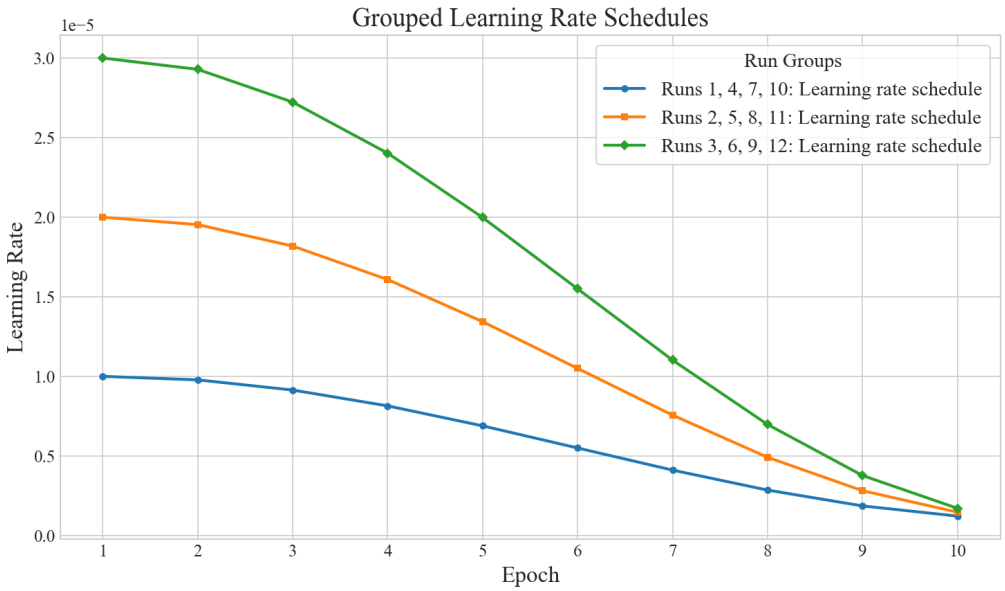
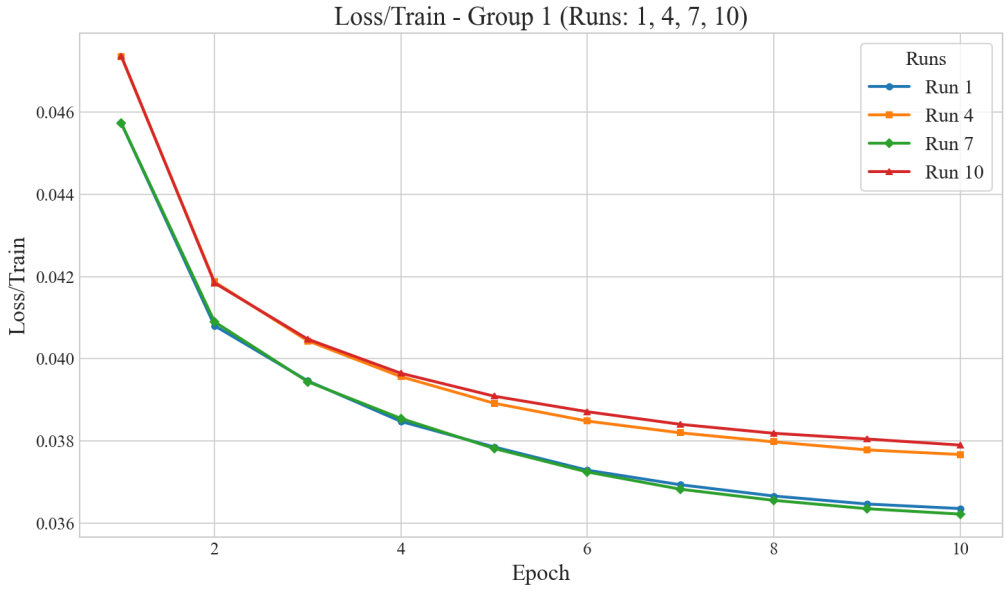


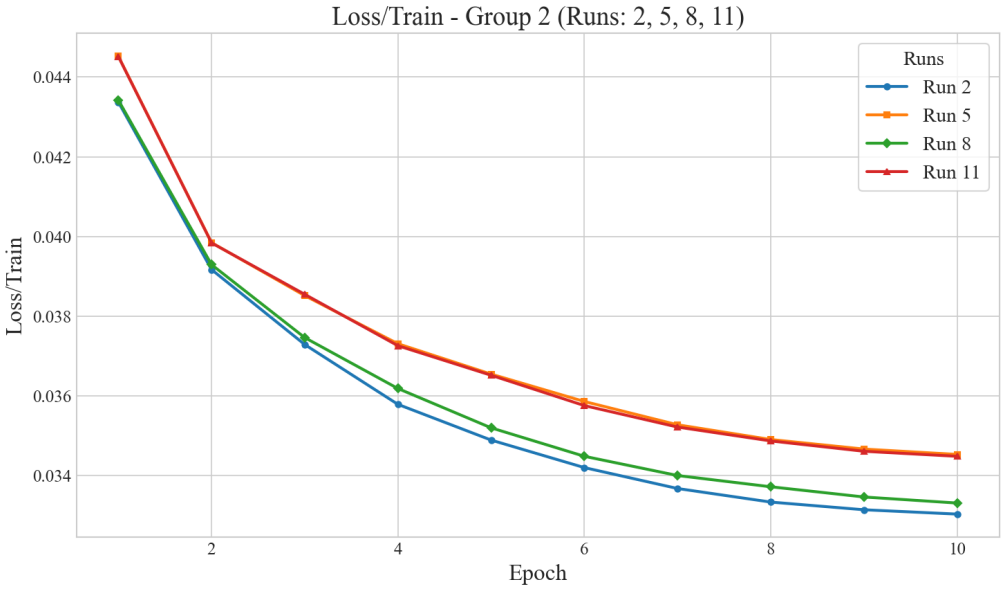
Figure 4.1: Grouped learning rate schedules for different runs. Each group shares the same learning rate progression.
Runs 1, 4, 7, 10 with initial learning rate of 10^{-5} coloured blue.
Runs 2, 5, 8, 11 with initial learning rate of $2{\cdot}10^{-5}$ coloured orange.
Runs 3, 6, 9, 12 with initial learning rate of $3{\cdot}10^{-5}$ coloured green.



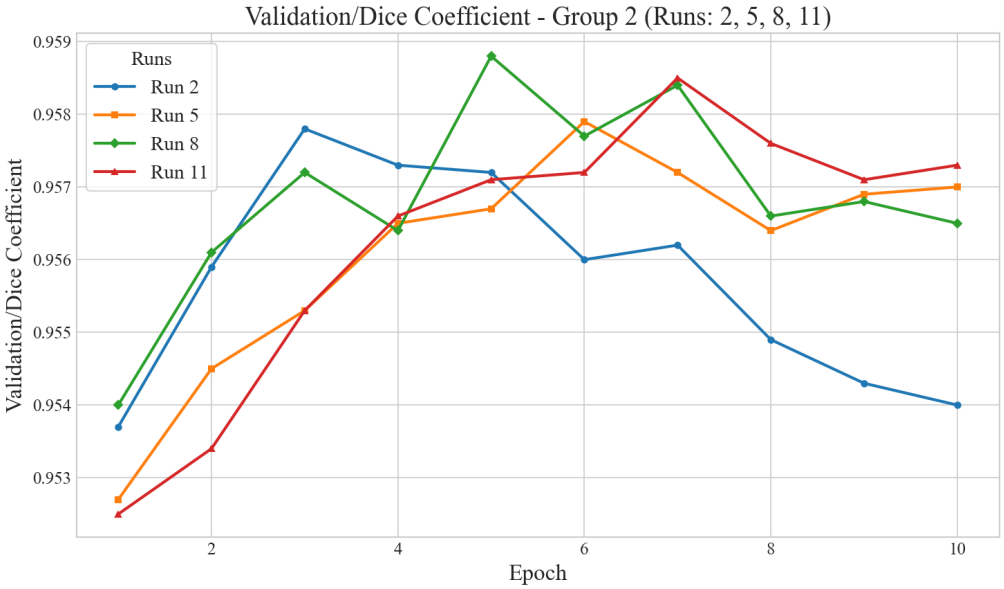
(a) Training Loss for Group 1 (Runs: 1, 4, 7, 10, with initial learning rate of 10^{-5}).

(b) Validation Dice Coefficient across 10 epochs. Run 7 consistently outperforms the others, with Run 4 and Run 10 showing similar performance.

Figure 4.2: Comparison of training and validation metrics for Learning Rate Group 1 (Runs 1, 4, 7, 10). Subfigure (a) shows the reduction in training loss over epochs, while subfigure (b) presents the improvement in segmentation performance as measured by the Dice Coefficient.

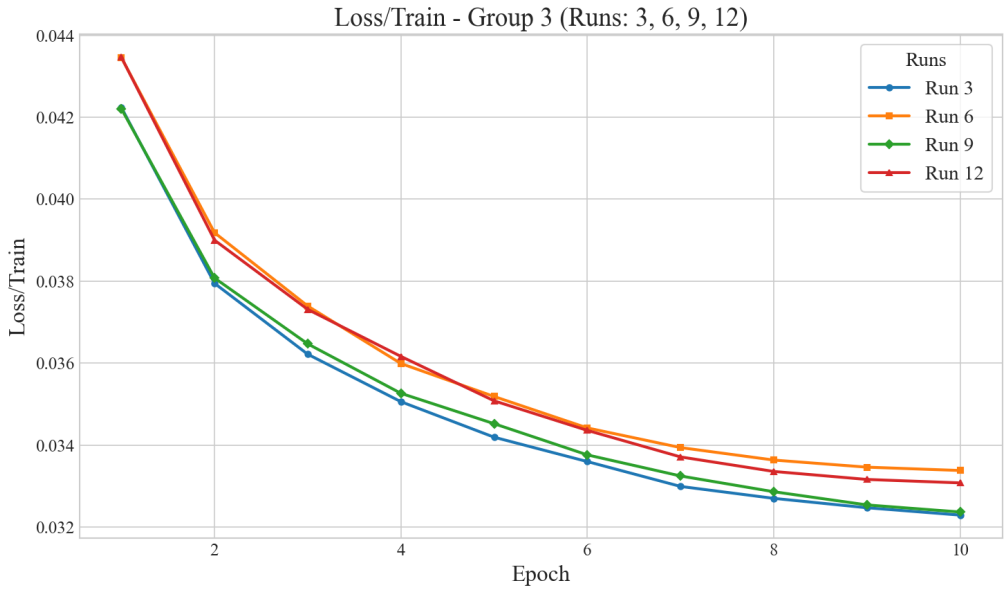


(a) Training Loss for Group 2 (Runs: 2, 5, 8, 11, with initial learning rate of $2{\cdot}10^{-5}$).

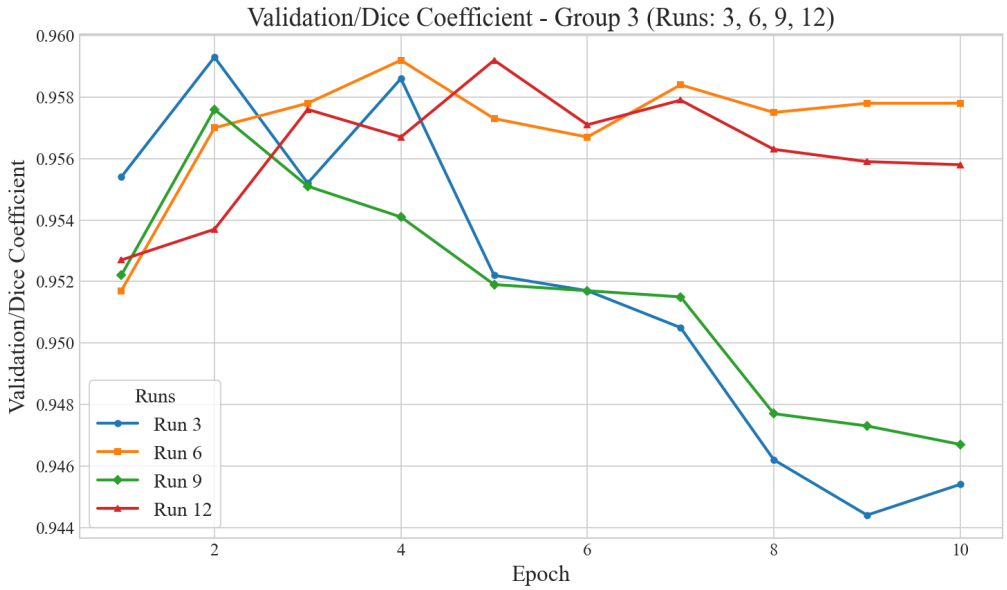


(b) Validation Dice Coefficient for Group 2 indicates highly fluctuating patterns with sharp rises and falls.

Figure 4.3: Metrics for Learning Rate Group 2: (a) shows the training loss over 10 epochs, indicating consistent convergence, while (b) illustrates the evolving segmentation accuracy.



(a) Training Loss for Group 3 (Runs: 3, 6, 9, 12, with initial learning rate of $3{\cdot}10^{-5}$).



(b) Validation Dice Coefficient for Group 3. Run 6 and 12 maintain a relatively steady improvement, while Runs 3 and 9 deteriorate and exhibit poor performance on the validation dataset.

Figure 4.4: Metrics for Learning Rate Group 3: (a) shows close training loss curves and (b) reveals *overfitting* in several runs.

The runs are grouped based on the learning rate, thus forming three groups, consisting of four runs each:

- Runs 1, 4, 7, 10 with initial learning rate of 10^{-5}
- Runs 2, 5, 8, 11 with initial learning rate of $2{\cdot}10^{-5}$
- Runs 3, 6, 9, 12 with initial learning rate of $3{\cdot}10^{-5}$

Group 1

Group 1 consists of the runs with an initial learning rate of 10^{-5}. These runs follow a consistent pattern in the training loss plot across epochs, with the loss function decreasing steadily and plateauing toward the final stages of training process.

The corresponding plot of the Dice Coefficient, computed at each epoch on the validation set, exhibits an overall increasing trend with minor fluctuations throughout the process.

The four runs within this group can naturally be divided into two subgroups based on batch size; runs 1 and 7 used a batch size of 4, while the remaining runs used a batch size of 8.

Overall, the plots suggest a robust and stable training process. Notably, the configurations using a batch size of 4 appear to slightly outperform those using a batch size of 8, as illustrated in the training and validation plots.

Group 2

Group 2 contains the runs initialized with a learning rate of $2 \cdot 10^{-5}$. The training loss curves for this group follow a similar trend to those in Group 1, though they reach lower final loss values by the end of training.

However, the Dice Coefficient plots show different behavior, exhibiting highly fluctuating patterns with sharp rises and falls. In most cases, the maximum Dice Coefficient is achieved around the middle stages of the training process.

These observations indicate that the learning rate in this group may be too high for stable training. In particular, Run 2 demonstrates poor validation behaviour, suggesting signs of *overfitting*.

Group 3

Group 3 includes the runs with an initial learning rate of $3 \cdot 10^{-5}$. The training loss for this group reaches even lower final values than those in the previous groups, indicating effective minimization of the objective function.

The Dice Coefficient curves reveal highly irregular and oscillatory behavior. The peak validation performance is typically achieved early in the training process, followed by a decreasing tendency and instability.

These results further support the hypothesis that a learning rate in this range may be excessively high. Among the runs in this group, Runs 6 and 12 - which used a batch size of 8 and weight decay values of 10^{-4} and 10^{-5} respectively - exhibited more stable and promising performance.

4.1.1 Optimization Results

The input parameters of the Automatic Mask Generator are determined with Hyperparameter Optimization and the objective function, as described in Section 3.3.6.

We are *optimizing* the value the Dice Coefficient value of the annotated images yielding the best values for the parameters as well as useful statistics (e.g., Figure 4.5, Figure 4.6).

4.1.2 Testing Results

In this section we are presenting the quantitative and qualitative results of the testing process. In order to compare our fine-tuned models we leverage the baseline models of SAM 2, namely the *small* and *large* checkpoints, thus setting a point of reference to better understand the results.

Evaluation Results

The Table 4.2 summarizes the quantitative outcomes of these experiments, illustrating the improvements achieved by the fine-tuned models relative to the base

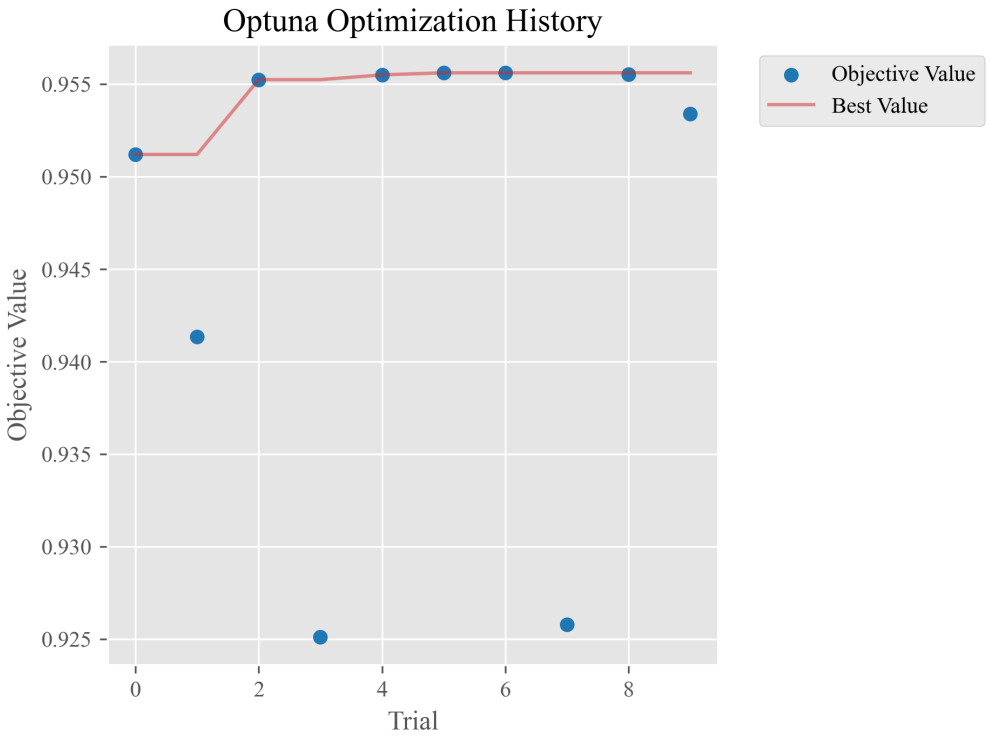


Figure 4.5: Example study for the Run 3. The best objective value found during the 10 trials is trial 5 with value 0.9551.

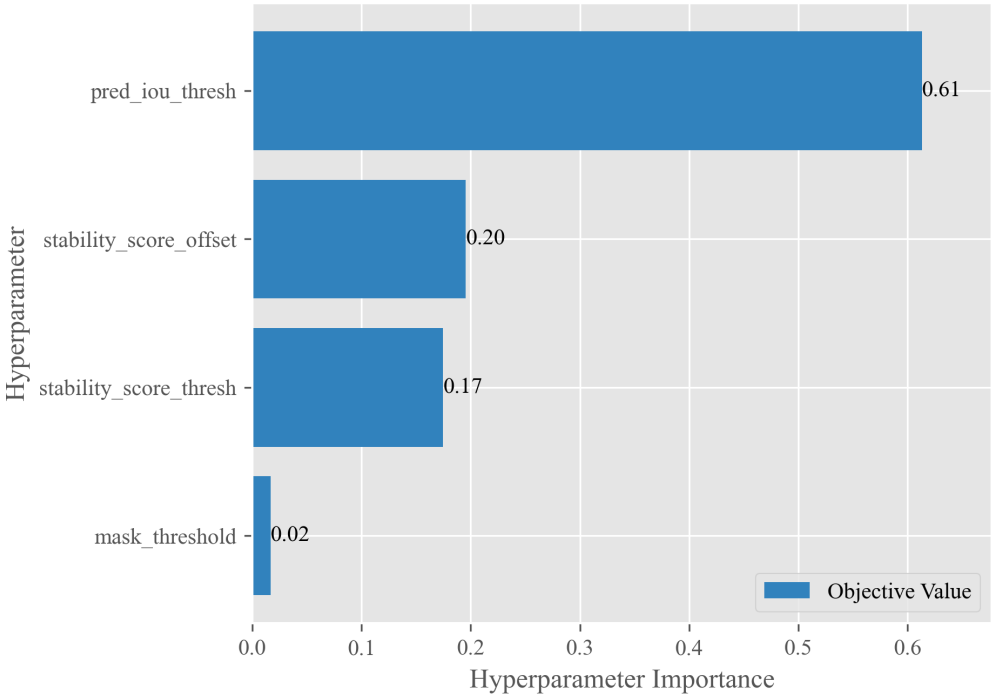


Figure 4.6: Example results of hyperaparameter importance based on Run 3. The Prediction IoU Threshold parameter is the most important, while Mask Threshold has minimal contribution in this configuration.

checkpoints, using the important metrics of IoU, Accuracy and Dice Coefficient.

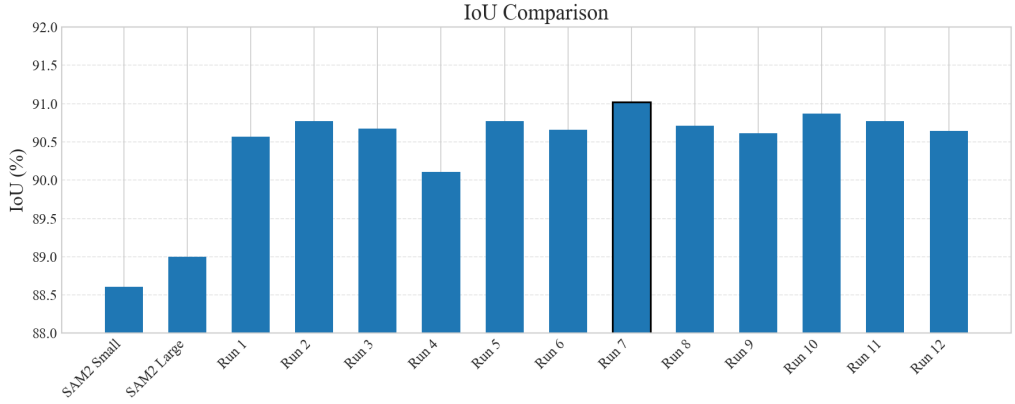
Table 4.2: Evaluation Results (IoU, Accuracy, Dice Coefficient, Recall) per Model.

Model	**IoU**	**Accuracy**	**Dice**	**Recall**
Base SAM 2 Small	88.61	91.12	93.82	92.01
Base SAM 2 Large	89.00	91.44	94.10	92.12
Run 1	90.57	92.60	95.00	94.60
Run 2	90.77	92.64	95.12	95.89
Run 3	90.67	92.54	95.06	96.02
Run 4	90.11	92.29	94.74	93.64
Run 5	90.77	92.70	95.12	95.22
Run 6	90.66	92.67	95.06	94.52
Run 7	**91.02**	**92.86**	**95.26**	**95.89**
Run 8	90.71	92.64	95.09	95.42
Run 9	90.61	92.49	95.03	95.96
Run 10	90.87	92.80	95.17	95.01
Run 11	90.77	92.68	95.12	95.49
Run 12	90.64	92.64	95.04	94.75

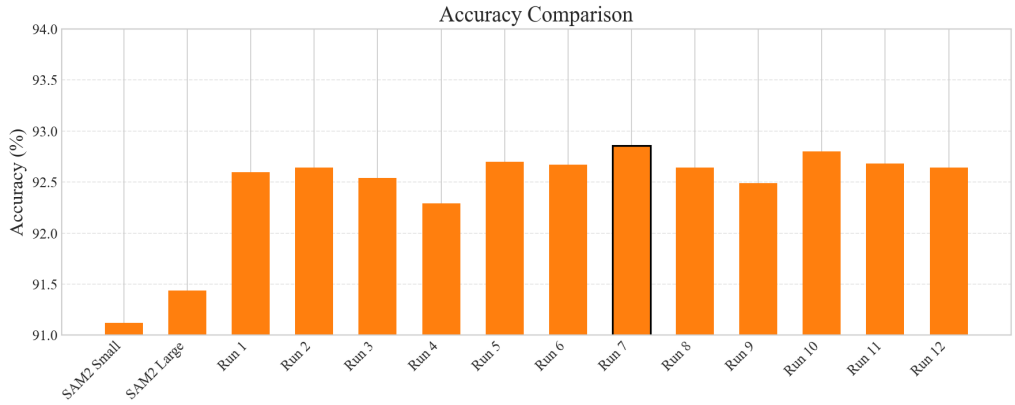
Run 7 was found to yield the best performance among all evaluated configurations. Consequently, whenever the fine-tuned model is mentioned in this thesis, it refers to the model from this run, and whenever the baseline model is mentioned, it refers to the *large* checkpoint, which outperforms the *small*.

Comparison Visualization

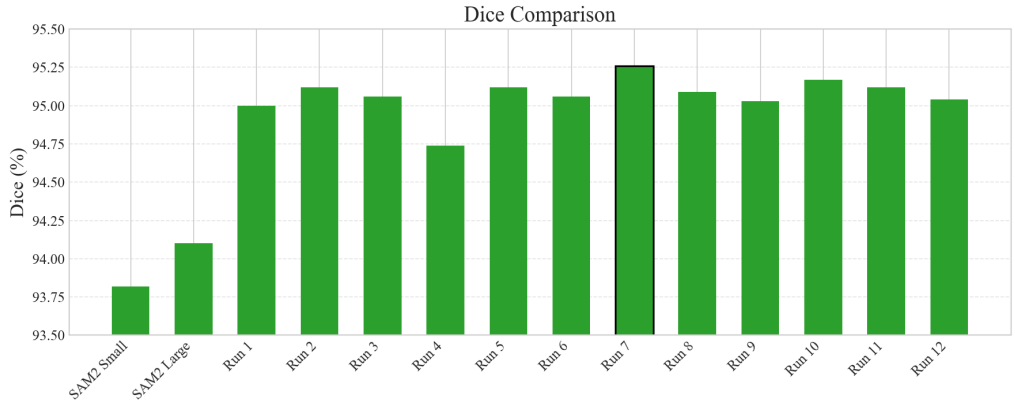
To better illustrate the differences between the baseline and fine-tuned model, we introduce a visual comparison highlighting how each model performs (with respect to the ground truth) in order to provide an intuitive understanding of the strengths and weaknesses of the two models.



(a) IoU comparison.



(b) Accuracy comparison.



(c) Dice Score comparison.

Figure 4.7: Barplots illustrating the quantitative results of the test configurations based on the IoU, Accuracy and Dice Score metrics between the two baseline and the fine-tuned models. The Run 7 is highlighted as it exhibits the best evaluation results.

In this visualization, each pixel is color-coded to reflect its classification: white pixels denote regions correctly segmented by both models; green pixels represent areas that the fine-tuned model successfully detected but were missed by the baseline; red pixels correspond to areas where the baseline succeeded but the fine-tuned model failed; and blue pixels indicate ground truth regions missed by both models.

We define three binary masks:

- M_{base}: predicted mask from the **baseline** model,
- $M_{\text{fine-tuned}}$: predicted mask from the **fine-tuned** model,
- M_{gt}: **ground truth** mask.

All masks are binarized such that a value of 1 corresponds to foreground (tesserae), and 0 corresponds to background.

We compute the following regions:

- **Both correct (white pixels):**

  $$R_{\text{both}} = M_{\text{base}} \cap M_{\text{fine-tuned}} \cap M_{\text{gt}}$$

  The region correctly segmented by both models.

- **Fine-tuned-only correct (green pixels):**

  $$R_{\text{fine-tuned-only}} = M_{\text{fine-tuned}} \cap M_{\text{gt}} \cap (\neg M_{\text{base}})$$

  The region correctly segmented by the fine-tuned but missed by the baseline model.

- **Baseline-only correct (red pixels):**

  $$R_{\text{base-only}} = M_{\text{base}} \cap M_{\text{gt}} \cap (\neg M_{\text{fine-tuned}})$$

  The region correctly segmented by the baseline but missed by the fine-tuned model.

- **Missed by both (blue pixels):**

  $$R_{\text{missed-both}} = M_{\text{gt}} \cap (\neg M_{\text{base}}) \cap (\neg M_{\text{fine-tuned}})$$

  Ground truth regions not detected by either model.

- **Background (black pixels):**

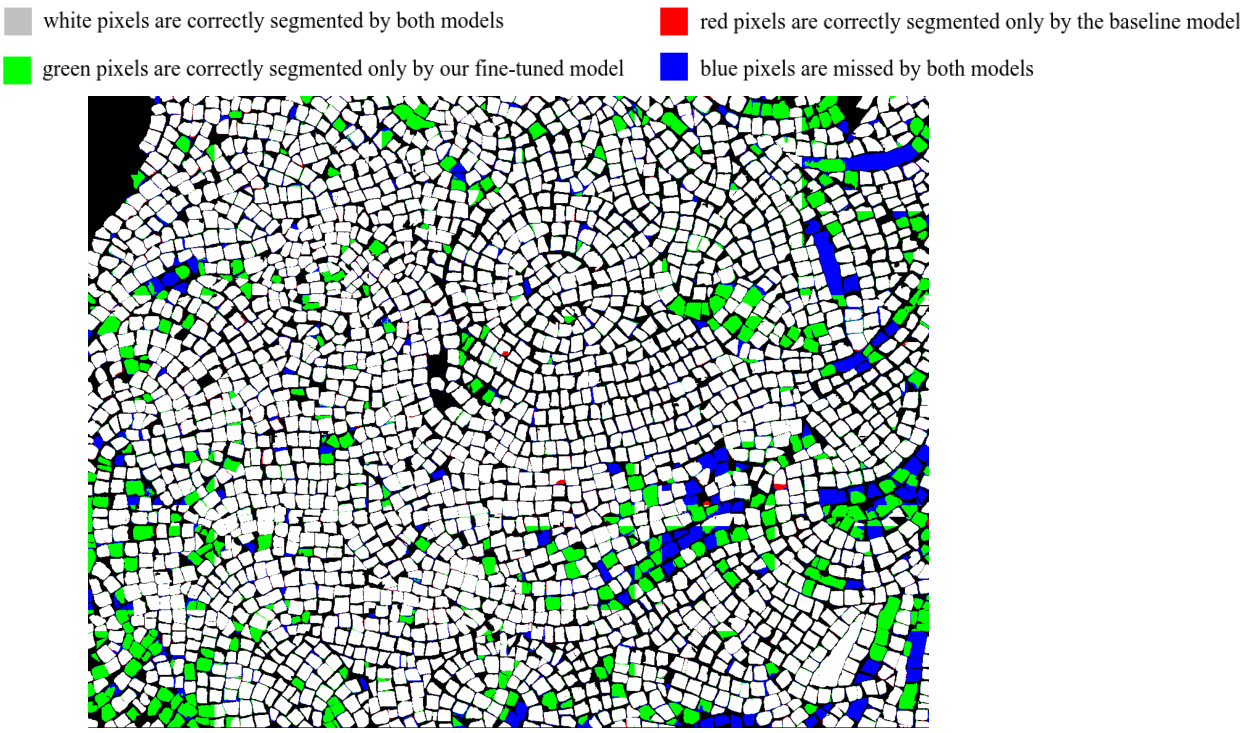
$$R_{\text{background}} = \neg\left(M_{\text{gt}}\right)$$

  Regions not belonging to the ground truth.

This visualization allows us to simultaneously highlight the following:

- Consistent correct segmentations (white),
- Improvements of the fine-tuned model (green),
- Regressions of the fine-tuned model (red),
- Missed ground truth areas (blue).

(a) Original sample image 1



(b) Segmentation comparison for sample image 1

Intersection over Union (IoU) scores:

Baseline model:	79.47%
Fine-tuned model:	84.96%

Figure 4.8: Comparison of original and segmentation results for sample image 1. In this illustration, white pixels are correctly segmented by both models, green are correctly segmented **only** by our fine-tuned model, red **only** by the baseline model, while blue pixels are missed by both. Black pixels represent the background.

4.2 Comparison with Existing Methods

To better comprehend the performance of our model and the general results of this thesis, we ought to compare this work with past works in the same field. As previously mentioned in Section 1.2, there are several major works alongside with the corresponding papers.

4.2.1 Past work based on the Watershed Algorithm (2008)

This work by Lamia Benyoussef and Stéphane Derrode is based on the Watershed Algorithm which is aided by a tesserae-based k-means classification (yielding higher accuracy than a pixel-based strategy). In the study of image processing, a watershed is a transformation defined on a grayscale image. The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges. It is one of the initial approaches and the authors also proposed a method to estimate the main directional guidelines of tesserae in mosaics. [9]

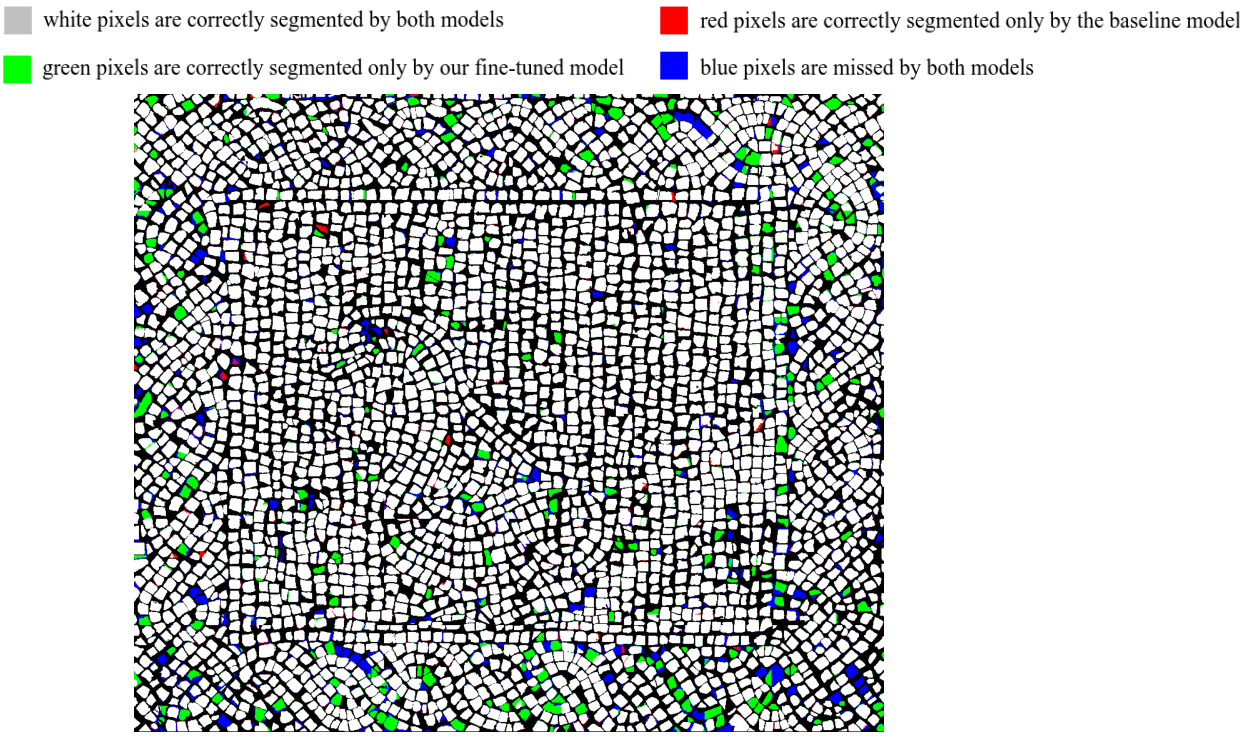
The researchers conducted a thorough work on the task and produced highly accurate results, with the guideline being mapped to the connecting glue in the real mosaics, thus employing a tesserae-based classification. The only downside of the method is that it was limited by the technology of the period approximately 20 years ago.

Modern approaches, such as ours, employ Deep Learning models which are initially trained to millions of sample images and are afterwards fine-tuned to fit the specific task, learning meaningful visual features directly from data.

4.2.2 Past works based on fine-tuning U-net (2020, 2021)

In more recent years, a remarkable work in 2020 proposed a Deep Learning technique based on a Convolutional Neural Network, U-net. The researchers trained the network on a set of annotated mosaic images and performed segmentation on pixel-level, which yielded impressive results. [11] Building upon this approach, in 2021, the Mo.Se. (Mosaic Segmentation) Algorithm was proposed, combin-

(a) Original sample image 2



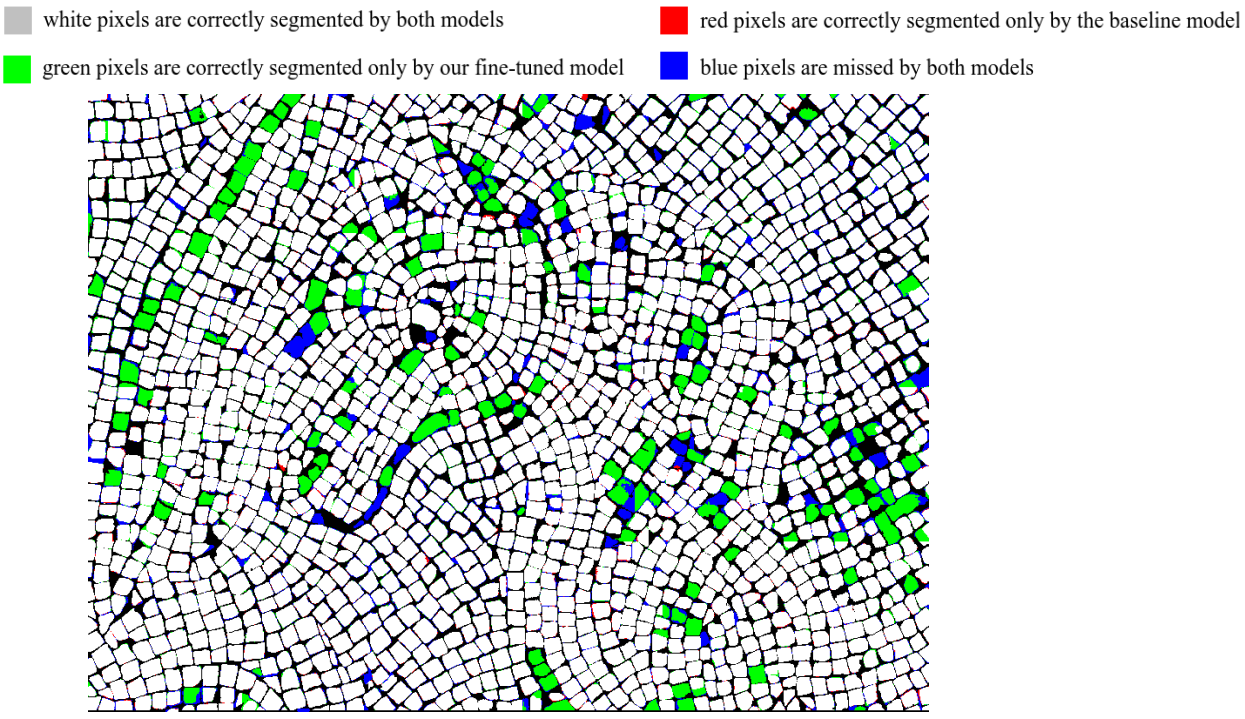
(b) Segmentation comparison for sample image 2

Intersection over Union (IoU) scores:

Baseline model:	84.41%
Fine-tuned model:	86.77%

Figure 4.9: Comparison of original and segmentation results for sample image 2. In this illustration, white pixels are correctly segmented by both models, green are correctly segmented **only** by our fine-tuned model, red **only** by the baseline model, while blue pixels are missed by both. Black pixels represent the background.

(a) Original sample image 3



(b) Segmentation comparison for sample image 3

Intersection over Union (IoU) scores:

Baseline model:	86.04%
Fine-tuned model:	91.02%

Figure 4.10: Comparison of original and segmentation results for sample image 3. In this illustration, white pixels are correctly segmented by both models, green are correctly segmented **only** by our fine-tuned model, red **only** by the baseline model, while blue pixels are missed by both. Black pixels represent the background.

ing the U-net 3 Network with the Watershed Algorithm. The authors themselves stated that the combination of learning-based methods with procedural ones enhances segmentation performance in terms of overall accuracy. [12]

These works can be compared more directly with our work; both works involved manual annotation and fine-tuning an existing powerful segmentation model, with the difference being on the choice of the model itself. The researchers of the past works provided useful statistics on the tesserae themselves, as obtained by the datasets that they used. These statistics included i.e. the average area of the tesserae. Finally, the implementation of the Watershed Algorithm alongside the Deep Learning foundation technique is not used in this work.

While these works mark important progress in the automatic segmentation of mosaics, our method offers several key improvements. By fine-tuning SAM 2, a state-of-the-art vision foundation model, we benefit from a much richer pre-learned representation. Furthermore, unlike U-net and its variants, which often require transfer learning on smaller datasets, SAM 2 already possesses robust zero-shot capabilities, allowing fine-tuning to converge faster and generalize better, which is vital due to the nature of the problem with mosaics being often damaged due to age.

4.2.3 Evaluation metrics comparison

In the paper *"Segmentation of Mosaic Images Based on Deformable Models Using Genetic Algorithms"* [10] the researchers proposed a universal dataset and a procedure in order to evaluate mosaic segmentation algorithms and compare them with past works. We explain them below for clarity.

Metrics

Let N_{gt} denote the number of ground truth tiles, and N_{pred} denote the number of predicted regions. For each ground truth tile i, let T_i represent the set of pixels belonging to that tile, and for each predicted region j, let R_j represent the set of pixels in that region.

For each ground truth tile T_i, let region_i be the predicted region R_j that maximizes

the size of the intersection $T_i \cap R_j$, i.e.,

$$\text{region}_i = \arg\max_j |T_i \cap R_j|.$$

Essentially, region_i is the predicted tile (**connected** region) that overlaps the most with the ground truth tile T_i.

The **Count Error** is defined as

$$\text{Count Error} = \frac{|N_{\text{gt}} - N_{\text{pred}}|}{N_{\text{gt}}}.$$

The **Precision** metric is computed as the average, over all ground truth tiles, of the fraction of pixels in the best matching predicted region that also belong to the ground truth tile:

$$\text{Precision} = \frac{1}{N_{\text{gt}}} \sum_{i=1}^{N_{\text{gt}}} \frac{|T_i \cap \text{region}_i|}{|\text{region}_i|},$$

where $|\text{region}_i|$ denotes the total number of pixels in the predicted region region_i.

The **Recall** metric is computed as the average, over all ground truth tiles, of the fraction of pixels in the ground truth tile that are correctly predicted in the best matching predicted region:

$$\text{Recall} = \frac{1}{N_{\text{gt}}} \sum_{i=1}^{N_{\text{gt}}} \frac{|T_i \cap \text{region}_i|}{|T_i|},$$

where $|T_i|$ is the total number of pixels in the ground truth tile T_i.

Finally, the **F-measure** is the harmonic mean of Precision and Recall:

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Test Dataset

The researchers kindly agreed to provide this dataset, although only a *subset* of the original 5 images was available. This subset consists of 2 images (*Museum with ID: 7* and *University with ID: 11*) and we will evaluate our model while adjusting the findings of the previous works based on this subset.

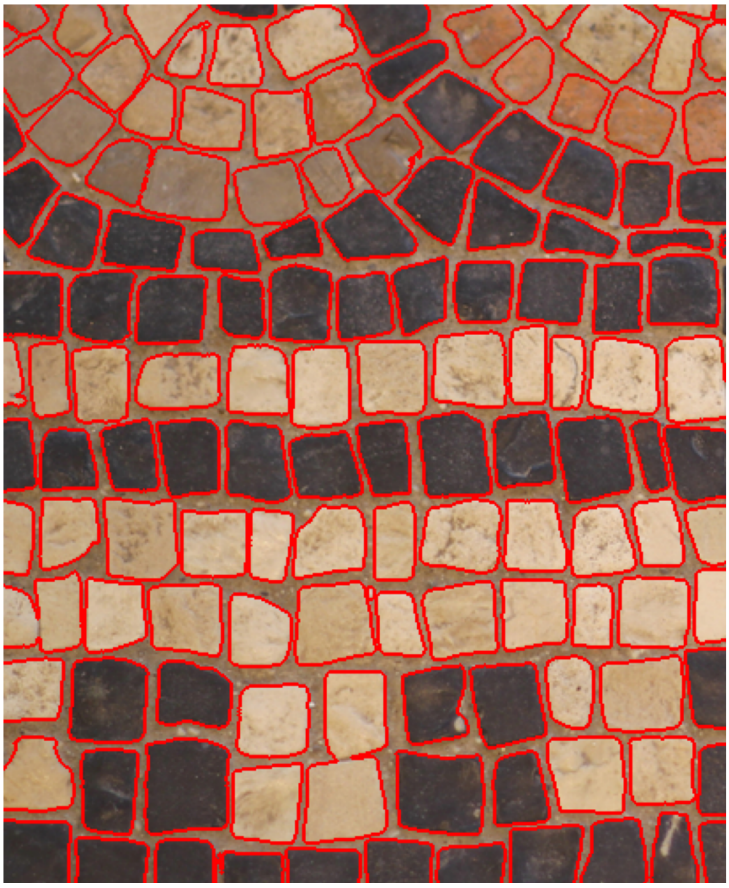


Figure 4.11: Segmentation of the *Museum* image with our method.

Table 4.3: Segmentation results on ImageIDs 7 and 11

ImageID	TOS				GA			
	Cnt	Prec	Rec	Fm	Cnt	Prec	Rec	Fm
7	0.14	64	87	74	0.03	50	76	60
11	0.90	63	78	70	0.03	46	67	55
Avg	**0.52**	**64**	**83**	**72**	**0.03**	**48**	**72**	**58**

ImageID	Unet				Unet3			
	Cnt	Prec	Rec	Fm	Cnt	Prec	Rec	Fm
7	0.30	65	80	73	25	75	90	82
11	0.29	65	72	69	14	71	83	77
Avg	**0.30**	**65**	**76**	**71**	**0.20**	**73**	**87**	**80**

ImageID	SAM 2 (large checkpoint)				Fine-Tuned SAM 2			
	Cnt	Prec	Rec	Fm	Cnt	Prec	Rec	Fm
7	12.38	62	75	68	0.01	76	81	78
11	1.52	83	87	85	0.03	89	86	87
Avg	**6.95**	**72**	**81**	**76**	**0.02**	**82**	**83**	**83**

Table 4.4: Segmentation results on the metrics of the *"Segmentation of Mosaic Images Based on Deformable Models Using Genetic Algorithms"* [10] paper researchers on the dataset we propose.

Image	**Our Method**			
	Cnt	Prec	Rec	Fm
Lizard	0.26	92.22	81.74	86.67
Horse	4.89	83.87	60.20	70.09
Floor Pattern 2	0.69	73.88	72.00	72.93
Duck	0.16	87.75	68.31	76.82
Tiger	1.08	76.47	66.76	71.28
Floor Pattern 1	0.33	94.37	69.91	80.32
Floor Pattern 3	0.13	91.80	81.92	86.58
Average	**1.08**	**85.77**	**71.55**	**77.81**

Discussion

5.1 Summary

This work experiments with the SAM 2 model by META AI and how it performs in the problem of automatic mosaic segmentation in the fine tesserae. These tesserae act as the regions of interest in the process of the segmentation and the main problem is that the mosaics are often old and damaged, making it difficult for the baseline models of SAM 2 to correctly identify them in depictions.

To tackle this problem and produce potentially better results than the original SAM 2, we applied a common and growing technique in the field of ML, which involves training an existing - usually foundation - model in a more specific dataset. This is often referred to as *transfer learning* and results in models improving their capabilities, *"tailored"* to a more niche task.

We used this process and achieved highly accurate segmentation on a test dataset of mosaics. Both visual comparisons and evaluation metrics illustrated a significant improvement in comparison to the baseline models. We provide a benchmark dataset[1], the accompanying code[2], and the fine-tuned SAM 2 model[3] to facilitate future comparisons and experiments on mosaic tesserae segmentation.

5.2 Comparison to Previous Works

The Table 4.3 demonstrates the performance comparison between previous methods and our work. We will mainly focus on the Count Error (Cnt) and F-measure (Fm) metrics.

The *Tessella-oriented segmentation* (TOS) [9] method has a very strong Fm value of 72%, albeit showing a weak Cnt value of 0.52. With this method being one of the first and most influential, while the researchers did not have access to recent technology, the results are very impressive.

The method based on *Genetic Algorithms* (GA) [10] lacks in the Fm metric, dis-

[1] https://huggingface.co/datasets/ckapelonis02/mosaics-images
[2] https://github.com/ckapelonis02/mosaic-sam2
[3] https://huggingface.co/ckapelonis02/SAM2-mosaic

playing a poor performance of 58%, yet achieves a Cnt of 0.03. Therefore, the number of predicted regions is very close to the number of ground truth tesserae. The low percentage of Fm suggests that these predicted regions might be misaligned or inaccurate.

The fine-tuned *Unet* [11] model results show a Cnt of 0.30 and Fm of 71% which is very impressive and proved that fine-tuning powerful segmentation models is the future of efficiently solving this task of mosaics segmentation.

Further proof of the above is given by the Mo.Se. *Unet3* [12] model. With a Cnt of 0.20 and Fm of 80%, Mo.Se. exhibits the most impressive results.

For our method, we fine-tuned the SAM 2 model and also applied to the resulting predictions **morphological opening** to remove small noise, **removed very small regions** with area less than a decided threshold and, finally, **erosion** to separate the tesserae very close to each other. Our fine-tuned SAM 2 model achieves a Cnt of 0.02 and Fm of 83%, indicating that our approach performs reliably on the given testing subset.

Beyond achieving strong metrics, our method demonstrates excellent generalization capabilities. The fine-tuned SAM 2 model handles mosaics of varying styles, tesserae sizes, and resolutions, maintaining reliable performance even on images that differ significantly from those in the training set. This robustness indicates that our approach can be effectively applied to diverse mosaic collections, making it a versatile tool for large-scale digitization, analysis in cultural heritage projects and even real-time segmentation scenarios.

5.3 Limitations and Outlook

Any serious work in the field of ML engineering involving the pipeline of training, validation and, finally, evaluation requires both computational power and also time commitment.

In this thesis, the resources provided by the platform Kaggle in order to train and evaluate the models were limited to 30 hours per week, while every session could not exceed 9 hours. At the same time, the GPU and RAM provided by the platform were not as powerful as advised by the META AI team in the training instructions.

A common problem included session interruptions due to network instability and

bugs from the platform itself. This resulted in many research hours wasted.

The manual annotation process was restricted to a small set of images because of time limitations set by the thesis scope, also not allowing the model to reach the full potential due to data scarcity through the training process.

Finally, while our primary comparison focused on fine-tuned versions of the SAM 2 model, a more comprehensive evaluation could include a wider range of segmentation models from both classical and deep learning approaches, and even edge detection-based methods. Incorporating additional baselines would provide a clearer picture of how SAM 2 performs relative to other state-of-the-art methods and highlight its strengths and limitations more thoroughly.

5.4 Future Work

Looking at this thesis, while the problem seems easy to understand, we encountered many unexpected problems throughout. The access to powerful resources would highly accelerate the process and produce significantly better results. If this could be achieved with a local setup, any network issues would not hinder the process as well. Ultimately, similarly to any ML problem, more computational power and time abundance could help produce more reliable results.

Additionally, in the future, extension to the work can include:

- **Including additional baseline methods for comparison.** Expanding the evaluation to incorporate other segmentation models and even classical edge detection-based approaches would provide a more comprehensive assessment of SAM 2's performance, highlighting its relative advantages and limitations.
- **Exploring alternative segmentation models with stronger transfer learning capabilities.** For instance, the model presented in the paper *"Highly Accurate Dichotomous Image Segmentation"* [55] aims to segment highly accurate objects from natural images and may be well-suited for this task, as it generates very detailed masks.
- **Expanding the mosaic dataset.** To tackle the problem of the limited annotated images, the dataset could be enriched to include a broader range of

mosaics, showcasing damage and erosion due to age, in order to improve the generalization abilities of the model in real-world mosaic images.

- **Adapting the SAM 2 architecture.** In order to optimally fit the given task, there could be adjustments to the existing architecture by modifying components of the SAM 2 model. This could potentially result in improved accuracy and robustness.
- **Refining post-processing techniques.** Further improvements could be made by designing smarter post-processing steps in order to adjust inaccurate segmentations by merging fragmented tesserae or better separating overlapping ones.
- **Comparing mosaic styles across different periods.** By analyzing the shapes, sizes, and arrangements of tesserae, it becomes possible to study stylistic differences between mosaics from various historical eras.
- **Modeling tesserae shapes mathematically.** Using tools from mathematical morphology, the contours and central points of tesserae can be quantitatively described, enabling precise shape comparisons.

5.5 Conclusion

In this thesis we studied the problem of segmenting mosaic images into the tesserae that compose them, with the aim of using the final segmentation to digitally replicate these mosaics, in collaboration with the Micromachining & Manufacturing Modeling Lab from the Department of Production Engineering & Management. This is a task that has already been studied by several researchers who primarily utilize Deep Learning methods to achieve optimal performances.

Our work involved fine-tuning the SAM 2 model with a custom training dataset and, eventually, evaluating it in contrast to the baseline models provided by the META AI research team. We optimized a set of parameters to best fit the Automatic Mask Generator optimally for every fine-tuned model on the test dataset.

The quantitative results, based on evaluation metrics, demonstrate a substantial improvement over the baseline SAM 2 model. Additionally, the qualitative results, such as visual comparisons on real mosaics, further confirm that our fine-tuned model outperforms previous approaches and offers a promising step toward automating tesserae segmentation.

Bibliography

[1] Luis Stephenson. "Cultural Heritage: Its Significance and Preserving." In: *Anthropology* 11.321 (2023). url: https://www.longdom.org/open-access/cultural-heritage-its-significance-and-preserving.pdf.

[2] E. Doria and F. Picchio. "Techniques for mosaics documentation through photogrammetry data acquisition. The Byzantine mosaics of the Nativity Church." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-2-2020 (2020), pp. 965–972. doi: 10.5194/isprs-annals-V-2-2020-965-2020. url: https://isprs-annals.copernicus.org/articles/V-2-2020/965/2020/.

[3] B. Zitová, J. Flusser, and F. Šroubek. "An application of image processing in the medieval mosaic conservation." In: *Pattern Analysis and Applications* 7.1 (2004), pp. 18–25.

[4] A. Zarghili et al. "Arabo-Moresque decor image retrieval system based on mosaic representations." In: *Journal of Cultural Heritage* 2.2 (2001), pp. 149–154.

[5] A. Zarghili, J. Kharroubi, and R. Benslimane. "Arabo-Moresque decor images retrieval system based on spatial relationships indexing." In: *Journal of Cultural Heritage* 9.3 (2008), pp. 317–325.

[6] Saining Xie and Zhuowen Tu. *Holistically-Nested Edge Detection*. 2015. arXiv: 1504.06375 [cs.CV]. url: https://arxiv.org/abs/1504.06375.

[7] Yu-Ichi Ohta, Takeo Kanade, and Toshiyuki Sakai. "Color information for region segmentation." In: *Computer Graphics and Image Processing* 13.3 (1980), pp. 222–241. issn: 0146-664X. doi: https://doi.org/10.1016/0146-664X(80)90047-7. url: https://www.sciencedirect.com/science/article/pii/0146664X80900477.

[8] Yuxuan Luo et al. "Concept-Level Semantic Transfer and Context-Level Distribution Modeling for Few-Shot Segmentation." In: *IEEE Transactions on Circuits and Systems for Video Technology* (2025). doi: 10.1109/TCSVT.2025.3554013.

[9] Lamia Benyoussef and Stéphane Derrode. "Tessella-oriented segmentation and guidelines estimation of ancient mosaic images." In: *Journal of Electronic Imaging* 17 (Oct. 2008). doi: 10.1117/1.3013543.

[10] Alberto Bartoli et al. "Segmentation of Mosaic Images Based on Deformable Models Using Genetic Algorithms." In: July 2016, pp. 233–242. isbn: 978-3-319-61948-4. doi: 10.1007/978-3-319-61949-1_25.

[11] Gianfranco Fenu et al. “Mosaic Images Segmentation using U-net.” In: Jan. 2020, pp. 485–492. doi: 10.5220/0008967404850492.

[12] Andrea Felicetti et al. “Mo.Se.: Mosaic image segmentation based on deep cascading learning.” In: *Virtual Archaeology Review* 12 (Jan. 2021), p. 25. doi: 10.4995/var.2021.14179.

[13] Cambridge Dictionary. *Artificial Intelligence*. 2025. url: https://dictionary.cambridge.org/dictionary/english/artificial-intelligence.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Global Edition. Pearson, 2020, pp. 19–22, 801–807.

[15] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997, p. xv. isbn: 0070428077.

[16] Daniel Kokotajlo et al. “AI 2027.” In: Available at: https://ai-2027.com/. Apr. 2025.

[17] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Online: Andriy Burkov, 2019.

[18] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Available at: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning. Cambridge University Press, 2014.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. isbn: 0262039249. doi: 10.5555/3312046.

[20] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2nd. Available at: http://szeliski.org/Book/. Springer, 2022.

[21] Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer Nature, 2023. isbn: 978-3-031-45467-7. doi: 10.1007/978-3-031-45468-4.

[22] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. url: https://arxiv.org/abs/1711.05101.

[23] Tom Fawcett. “An introduction to ROC analysis.” In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861–874. issn: 0167-8655. doi: https://doi.org/10.1016/j.patrec.2005.10.010. url: https://www.sciencedirect.com/science/article/pii/S016786550500303X.

[24] Li Yang and Abdallah Shami. "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice." In: *CoRR* abs/2007.15745 (2020). arXiv: 2007.15745. url: https://arxiv.org/abs/2007.15745.

[25] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. url: https://arxiv.org/abs/1706.03762.

[26] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG]. url: https://arxiv.org/abs/2108.07258.

[27] Irwin Sobel. "An Isotropic 3x3 Image Gradient Operator." In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[28] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. "Semantic texton forests for image categorization and segmentation." In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8. doi: 10.1109/CVPR.2008.4587503.

[29] C. Lindner et al. "Fully Automatic Segmentation of the Proximal Femur Using Random Forest Regression Voting." In: *IEEE Transactions on Medical Imaging* 32.8 (2013), pp. 1462–1472. doi: 10.1109/TMI.2013.2258030.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. url: http://arxiv.org/abs/1411.4038.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. url: https://arxiv.org/abs/1505.04597.

[32] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV]. url: https://arxiv.org/abs/1606.00915.

[33] Ying Yu et al. "Techniques and Challenges of Image Segmentation: A Review." In: *Electronics* 12.5 (2023). issn: 2079-9292. doi: 10.3390/electronics12051199. url: https://www.mdpi.com/2079-9292/12/5/1199.

[34] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. url: https://arxiv.org/abs/2010.11929.

[35] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. url: https://arxiv.org/abs/2304.02643.

[36] Nikhila Ravi et al. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV]. url: https://arxiv.org/abs/2408.00714.

[37] L. Vincent and P. Soille. "Watersheds in digital spaces: an efficient algorithm based on immersion simulations." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (1991), pp. 583–598. doi: 10.1109/34.87344.

[38] Jack Bell et al. *The Future of Continual Learning in the Era of Foundation Models: Three Key Directions*. 2025. arXiv: 2506.03320 [cs.LG]. url: https://arxiv.org/abs/2506.03320.

[39] Zhang Jiaxing and Tang Hao. *SAM2 for Image and Video Segmentation: A Comprehensive Survey*. 2025. arXiv: 2503.12781 [cs.CV]. url: https://arxiv.org/abs/2503.12781.

[40] Tianrun Chen et al. *SAM2-Adapter: Evaluating Adapting Segment Anything 2 in Downstream Tasks: Camouflage, Shadow, Medical Image Segmentation, and More*. 2024. arXiv: 2408.04579 [cs.CV]. url: https://arxiv.org/abs/2408.04579.

[41] Zhiling Yan et al. *Biomedical SAM 2: Segment Anything in Biomedical Images and Videos*. 2024. arXiv: 2408.03286 [cs.CV]. url: https://arxiv.org/abs/2408.03286.

[42] Mingya Zhang et al. *Path-SAM2: Transfer SAM2 for digital pathology semantic segmentation*. 2024. arXiv: 2408.03651 [eess.IV]. url: https://arxiv.org/abs/2408.03651.

[43] Jiayuan Zhu et al. *Medical SAM 2: Segment medical images as video via Segment Anything Model 2*. 2024. arXiv: 2408.00874 [cs.CV]. url: https://arxiv.org/abs/2408.00874.

[44] Shruti Jadon. "A survey of loss functions for semantic segmentation." In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, Oct. 2020, pp. 1–7. doi: 10.1109/cibcb48159.2020.9277638. url: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638.

[45] Jun Ma. *Segmentation Loss Odyssey*. 2020. arXiv: 2005.13449 [eess.IV]. url: https://arxiv.org/abs/2005.13449.

[46] Carole H. Sudre et al. "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations." In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer International Publishing, 2017, pp. 240–248. isbn: 9783319675589. doi: 10.1007/978-3-319-67558-9_28. url: http://dx.doi.org/10.1007/978-3-319-67558-9_28.

[47] Fei Zhu et al. *Rethinking Confidence Calibration for Failure Prediction*. 2023. arXiv: 2303.02970 [cs.LG]. url: https://arxiv.org/abs/2303.02970.

[48] Fabian Kuppers et al. "Multivariate Confidence Calibration for Object Detection." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2020.

[49] Rabindra Shaw. *Artificial Intelligence for Future Generation Robotics*. June 2021. isbn: 978-0-323-85498-6. doi: 10.1016/C2020-0-02655-1.

[50] S. Mishra et al. *Cognitive Big Data Intelligence with a Metaheuristic Approach*. Cognitive Data Science in Sustainable Computing. Academic Press, 2021. isbn: 9780323851183. url: https://books.google.gr/books?id=EBU3EAAAQBAJ.

[51] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG]. url: https://arxiv.org/abs/1608.03983.

[52] Ibrahem Kandel and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset." In: *ICT Express* 6.4 (2020), pp. 312–315. issn: 2405-9595. doi: https://doi.org/10.1016/j.icte.2020.04.010. url: https://www.sciencedirect.com/science/article/pii/S2405959519303455.

[53] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. arXiv: 1206.2944 [stat.ML]. url: https://arxiv.org/abs/1206.2944.

[54] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML]. url: https://arxiv.org/abs/1807.02811.

[55] Xuebin Qin et al. "Highly Accurate Dichotomous Image Segmentation." In: *ECCV*. 2022.